**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 **Ten Core Security Principles**

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1.      ValidateInput Data | One of the first important steps of any system that utilizes input is to never trust the input data. It is essential to validate all data before use, to properly allow/deny interactions with the system. Input validation helps prevent malicious errors from reaching sensitive code that’ll ultimately allow access to important information. |
| 2.      Heed Compiler Warnings | Compiler warnings help alert the developer to problems in the code before it is ever executed in run time. Warnings can help find unintended behavior or simply forgetting to initialize a variable or return from a function. Ultimately, helping the developer to clean up the code before execution. |
| 3.      Architect and Design for Security Policies | It’s important for security policies to be consistent and auditable by security teams. Also, it’s vital for the security implementations to define a proper threat, levels of data safety, and to be easily implemented or removed/upgraded as the scope changes. |
| 4.      Keep It Simple | Complex code has more potential to hide bugs and incorrect implementations. Industry standard coding practices ensure easy clear implementation that makes the code readable. This ultimately reduces the chance for logical errors and makes the code more scalable for future developers. |
| 5.      Default Deny | By default, it’s important for systems to be strict and only allow explicit access to the system or operations. This ensures proper baseline security for the system. |
| 6.      Adhere to the Principle of Least Privilege | The principle of least privilege grants the most minimum and necessary capabilities to users to complete their job at hand. This reduces the potential for compromises by avoiding unnecessary privileges. |
| 7.      Sanitize Data Sent to Other Systems | When data crosses boundaries it’s important that the data is properly handled to protect the data as well as ensure it meets the standards and expectations of the incoming system. This helps protect the system from injection attacks which can ultimately compromise the database. |
| 8.      Practice Defense in Depth | Using multiple controls layered reduces the risk that a single bug/malicious attack leads to a full system compromise/failure. |
| 9.      Use Effective Quality Assurance Techniques | Code reviews and testing, as well as automatic checks of potential risks and sensitive data are crucial to remediating the natural inclination of human error. This helps to ensure a more trustworthy product and making developers more confident on if there code is proper and secure. |
| 10.  Adopt a Secure Coding Standard | Usings a documented coding standard tailored to that specific language helps create a clear concise rubric on what to do and not do. This helps developers keep code simple and stay within scope to reduce risky coding practices. |

### 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** | **Choosing the correct types helps avoid truncation, sign errors, or other behaviors that can create vulnerabilities in the logic. Using fixed-size types when able and avoiding implicit conversions help avoid this.** |

| **Noncompliant Code** |
| --- |
| **Unsinged implicit conversion truncates causing unforeseen sign/size changes** |
| **Int wrongConversion(){**  **unsinged int u = 8000000000u; // unsinged int of large size to be converted**  **int x = u;                                         // narrowed conversion of unsinged to signed int**  **return x;**  **}** |

| **Compliant Code** |
| --- |
| **Use of explicit, safe types and check ranges before conversion to ensure proper conversion** |
| **Int correctConversion(){**  **uint64\_t = 8000000000ull;**  **if (u > static\_cast,uint64\_t>(std::numeric\_limits<int32\_t>::max())){ //overflow handling making sure no truncation**  **return -1;**  **}**  **Return static\_cast<int32\_t>(u);**  **}** |

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

| **Principles(s):** 1,2,4,10 – Validating input and heeding compiler warnings to prevent truncation or overflow, keep code simple helps reduce errors |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.18 | typeConversionLoss | Detects implicit conversion, that can lose data |

**Coding Standard 2**

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | **STD-002-CPP** | **Values are to be checked for expected ranges and types. Logical errors like divide by zero can happen when the data values aren’t checked before comparison** |

| **Noncompliant Code** |
| --- |
| **No validation, so the function can divide by zero or assign a  value that is out of range of int** |
| **Int getAverage(int num1, int num2){**  **Return num1 / num2; //crashes if divided by zero or out of range assignment**  **}** |

| **Compliant Code** |
| --- |
| **Check input value and handle it properly** |
| **Std::optional<int> getAverage(int num1, int num2){ // optional used to safely return a nullopt if divide by zero occurs**  **If(num2 == 0) return std:;nullopt; //divide by zero found, return nullopt**  **Return num1 / num2;**  **}** |

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

| **Principles(s):** 1,6,9 – Validating input ensures value safety, least privilege prevents misuse, QA detects divide by zero |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 25.1 | S3518 (Division by zero) | Warning when operations risk divide by zero |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | **STD-003-CPP** | **Avoid buffer overflows by being careful with bound checking and string API calls** |

| **Noncompliant Code** |
| --- |
| **Buffer overflow by copying string, if it’s larger than 31 chars it will cause issues** |
| **Void wrongCopy (const char\* str\_input){**  **Char str\_copy**  **Std::strcpy(str\_copy, str\_input); // overflows when the input is too large**  **}** |

| **Compliant Code** |
| --- |
| **Use of std:string and bounded length checks ensures compliance** |
| **Void correctCopy(const std:: string& str\_input){**  **Std::string str\_copy = str\_input;  //properly manages length, truncates if over length**  **}** |

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

| **Principles(s):** 1,7,8 – Input validation and sanitization protect against buffer overflows, DiD ensures redundant checks across layers |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.18 | Buffer\_overflow | Flags unsafe string truncations and missing bound checks |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | **STD-004-CPP** | **Taking SQL queries without validating for injections can lead to access to sensitive information making an entire database vulnerable. It is important to utilize some type of parameters or prepared statement to check for SQL injection in queries.** |

| **Noncompliant Code** |
| --- |
| **Non validated query using a user input** |
| **std:: string userQuery = “SELECT \* FROM users WHERE name = ‘” + userInput + “’”;**  **db.execute(query);  // Injection if userInput contains some type of SQL injection statements like 1=1** |

| **Compliant Code** |
| --- |
| **Use of prepared statement ensures only validated queries can be executed in the db** |
| **Auto userQuery = db.prepate(“SELECT \* FROM users WHERE name = ?”);**  **userQuery.bind(1, userInput); //binds with prepared statement to ensure only valid queries are allowed to execute**  **userQuery.execute();** |

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

| **Principles(s):** 1,5,7,8 – Default-deny and sanitization enforce strict query rules, DiD helps to prevent DB compromise |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 25.1 | S3649 (SQL Injection) | Detects concatenated SQL strings and recommends a prepared statement |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | **STD-005-CPP** | **Managing memory errors are crucial to avoid major exploitations or vulnerabilities, utilizing smart pointers are a good way to handle the utilization of memory in C++** |

| **Noncompliant Code** |
| --- |
| **Use-after-free risk since the pointer is assigned after being deallocated in memory** |
| **Char\* p = “HELLO”;**  **Std::free(p);**  **\*p = ‘a’;  //use-after-free, vulnerable to exploits and major issues** |

| **Compliant Code** |
| --- |
| **Use of smart pointers alleviates need to delete since memory is automatically freed when pointer goes out of scope** |
| **auto ptr = std::make\_unique<int>(10);**  **// no delete needed since smart pointer**  **\*ptr = 40;** |

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

| **Principles(s):** 4,8,9,10 – Simplified design and secure coding standards reduce pointer errors, DiD and QA test and validate memory safety |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Possible | High | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 22.0 | LeakSanitizer | Run-time memory leak detector |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | **STD-006-CPP** | **Used for internal validations such as validating logic or invariations in the code** |

| **Noncompliant Code** |
| --- |
| **Using assert to validate user input** |
| **Void myFunction(int num1) {**  **Assert(num1 > 0); // NOT TO BE USED TO VALIDATE USER INPUT**  **}** |

| **Compliant Code** |
| --- |
| **Proper use to report internal logical errors or invariations** |
| **Void myCorrectFunction(int num1){**  **Int x = 9;**  **x = num1;**  **assert(x == num1); // used to check internal logic / invaritaion**  **}** |

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

| **Principles(s):** 2,4,9 – Heeding compiler warnings and ensuring simple logic ensures assertions are used correctly, QA helps validate internal logic |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 25.10 | S3908 (Assertions for debug only) | Ensures assertions aren’t used for runtime input validation |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | **STD-007-CPP** | **The use of exceptions is handy to catch errors and handle them gracefully.** |

| **Noncompliant Code** |
| --- |
| **The use of exceptions is handy to catch errors and handle them gracefully.**  **Cathcing exception and continuing to process in this bad state** |
| **try {**  **exceptionThrowingFunc();**  **} catch (Exception e&){**  **// catch the exception and continue with function in this unknown state rather than handle the error**  **}** |

| **Compliant Code** |
| --- |
| **Caught specific exceptions, logged it** |
| **try {**  **exceptionThrowingFunc();**  **} catch (Exception e&){**  **Std::cerr << “Error: “ << e.what() << std::endl; //error caught and logged**  **logError(e.what());**  **}** |

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

| **Principles(s):** 4,8,9 – Simplified, well-handled exceptions support DiD and reliable QA |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.18 | Exception Safety | Reports improper exceptions |
| SonarQube | 25.10 | S3984 (Catch generic exceptions) | Reports improper or empty catch blocks |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Commenting & Documentation** | **Commenting & Documentation** | **Clear comments help prevent functions from being misunderstood, incentivizing future developers to easily understand and work within their code base.** |

| **Noncompliant Code** |
| --- |
| **No explanation of code creates assumptions that lead to confusion** |
| **Int calcResult(int inputNum){**  **Return inputNum \* 78; // what is 78 and why is it important or being used, what does this function do**  **}** |

| **Compliant Code** |
| --- |
| **Code commenting explains the function and why it is being used and internal code** |
| **// Multiplies by 78 which is the applications scaling factor for later calculations**  **Int calcScale(int inputNum){**  **Return x \* 78;**  **}** |

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

| **Principles(s):** 3,4,9 – Secure and simple principles depend on clear documentation for consistency and audability |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 25.10 | S125 (Sections of code should no be commented out) | Enforces consistent commenting and documentation standards |

**Coding Standard 9**

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Consistent Naming Conventions** | **STD-009-CCP** | **Inconsistent names leads to confusion and errors, standardized naming within a code base enables easier recognizable code and reduces logical mistakes** |

| **Noncompliant Code** |
| --- |
| **Simple variables that will be used within the code named not what its purpose is leaves room for assumptions** |
| **Int x = userInput;**  **Int y = 35;**  **Int z = x + y; //what are these x y z variables achieving? Name them according to their objective** |

| **Compliant Code** |
| --- |
| **Use of a clear consistent style creates an easier reading and understanding** |
| **Int itemCount = userInput;**  **Int prevCount = 50;**  **Int totalCount = itemCount + prevCount; // Now we see that it is totaling up the count of two different variables** |

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

| **Principles(s):3, 4, 9 -** Consistency and simplicity in naming aids maintainability and QA code review |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Possible | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 25.10 | S117 (Variable names should comply with naming conventions) | Validates identifier naming consistency |

**Coding Standard 10**

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Hard coding credentials or logic can create a system with vulnerabilities and less scalable** | **STD-018-CPP** | **Hard coding credentials or logic can create a system with vulnerabilities and less scalable** |

| **Noncompliant Code** |
| --- |
| **Hard coding credentials or logic can create a system with vulnerabilities and less scalable**  **Store passwords in plain text in code** |
| **Std::string dbPassword = “password123”;** |

| **Compliant Code** |
| --- |
| **Gets code from config file or environment variables, not within code to keep the plain text view secure but still readable** |
| **Gets code from config file or environment variables, not within code to keep the plain text view secure but still readable**  **Std::string dbPassword = getEnv(“DB\_PASSWORD”);** |

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

| **Principles(s):** 5,6, 10 - Default-deny, least privilege, and secure standards prevent exposure of sensitive credentials |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | Possible | Medium | Critical | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 25.10 | S2068 (Hardcoded credentials) | Detects hardcoded secrets or credentials |

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 for compliance enforcement occurs throughout Green Pace’s DevSecOps pipeline. Security starts at the assessment and planning stage to ensure regulations and compliance is thought of at the beginning and into the design stage. Then Security scanning tools like Cppcheck & SonarQube are used for static analysis throughout the build and verify/test stage before merging to the repo. In the verify and test stage if vulnerability or violations exceed the threshold then the build is a failure and redesigned. Other checks like dependency checks help validate that required dependencies and libraries are also free of vulnerabilities that can create external vulnerabilities. All security implementations and reports are logged, and system monitoring is continuously performed with auditing and event alerts. Automation ensures ongoing compliance of coding standards while minimizing the manual effort, further enabling developers to focus on coding while still maintaining

a feedback loop.

### 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 | Possible | Low | High | 3 |
| STD-003-CPP | High | Likely | Medium | Critical | 4 |
| STD-004-CPP | Critical | Likely | Medium | Critical | 5 |
| STD-005-CPP | High | Possible | Medium | High | 4 |
| STD-006-CPP | Medium | Unlikely | High | Medium | 2 |
| STD-007-CPP | Medium | Possible | Low | Medium | 3 |
| STD-008-CPP | Low | Likely | Low | Medium | 2 |
| STD-009-CPP | Low | Possible | Low | Low | 1 |
| STD-010-CPP | Critical | Possible | Medium | Medium | 5 |

### 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 | Protects data when it is stored, using encryption standards like AES-256. Ensure that if that media is stolen or data infiltrated off a server, it remains confidential due to its encryption. To be applied for all data stores |
| Encryption in flight | Transmitted data is secured typically with encrypted tunnels like TLS or IPsec. This prevents bad actors from snooping or altering the data during communications. Useful for all forms of network traffic |
| Encryption in use | Use: Protects data while it is being processed by memory, either by hardware-based execution environments or software based. This allows applications to perform sensitive functions and operations without exposing the data in clear text. To be used during sensitive operations to ensure confidentiality. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Verifies users’ identity using a combination of user login information like. MFA, password, biometrics, tokens, etc. Used to protect access for all applications within the system, to prevent unauthorized access/entry into the system |
| Authorization | Least privilege and role-based access controls, only give access to parts of the system are necessary for said users to perform their tasks at hand. Limits access to limit possible damage such as changes to the database and addition of new users (or account takeover). Prevents privilege escalation and cyberattack techniques like lateral movement, which allows attackers to expand their control across an entire network. |
| Accounting | Provides auditing abilities of all actions on a system such as login attempts, DB changes, files accessed, etc. To be monitored and reviewed to trace incidents and respond properly. |

**\***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 |  |
| 2.0 | 09/19/2025 | Milestone | Zachary Hancock |  |
| 3.0 | 10/07/2025 | Project One | Zachary Hancock |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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