**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 | Ensure all user input is checked for errors, formatting, and malicious content. Be aware of all common injection attacks such as SQL injection and buffer overflows. Always validate and sanitize user inputs to prevent vulnerabilities. |
| 1. Heed Compiler Warnings | Treat compiler warnings as errors and fix them whenever possible. Compiler warnings can indicate potentially unsafe code that can provide attack vectors to malicious actors. |
| 1. Architect and Design for Security Policies | Integrate security policies into the architecture and design process when building software. Planning proactively ensures the team is on the same page from the start and implementing security as a first-class feature is better than fixing it later. |
| 1. Keep It Simple | More than just a security principle, but a principle that will surely benefit the security of the application nonetheless. The more complexity in a code base, the more opportunity for bugs creeping into the code and inviting vulnerabilities. |
| 1. Default Deny | Applications should be developed in a way that requires users to get explicit access as opposed to granting access to everyone and needing to revoke access. Whitelist, don’t blacklist. |
| 1. Adhere to the Principle of Least Privilege | Provide users with the minimum privileges needed to complete their tasks. This both prevents attacks and prevents them from spreading and becoming larger than they would have been. |
| 1. Sanitize Data Sent to Other Systems | Inspect data closely before sending it to other systems. A common example is taking user inputs and then sending them to a database system. This user input needs to be escaped to prevent SQL injections. |
| 1. Practice Defense in Depth | Implement multiple layers of security. Security policies should be implemented at every level of an organization including physical, network, and software. |
| 1. Use Effective Quality Assurance Techniques | Write tests for all code. Have code reviews before pushing code to production environments. Use testing practices such as unit testing, regression testing, and take advantage of security frameworks to detect potential vulnerabilities in a code base. |
| 1. Adopt a Secure Coding Standard | Utilize an existing secure coding standard to avoid common vulnerabilities. |

### 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 | Prevents overflow and underflow by using the correct data types. |

| **Noncompliant Code** |
| --- |
| Using int for array size risks overflow and undefined behavior. |
| int size = 10;  int arr[size]; |

| **Compliant Code** |
| --- |
| The size\_t data type ensures the correct number of bytes as it adapts to the target platform. |
| size\_t size = 10;  int\* arr = new int[size]; |

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

| **Principles(s):** The rule in SEI CERT that is associated with this standard is INT01-C. This rule states that you must use rsize\_t or size\_t for all integer values representi |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE | 1.0 | CertC-INT01 | Detects violations of this rule. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Prevents out-of-bounds indexing of arrays which gives unwarranted access to memory. |

| **Noncompliant Code** |
| --- |
| Failing to validate the value of an integer risks overflow when using the value to index an array. |
| T get\_value(int index) {  int arr[5] = { 0 };  T value = arr[index];  return value;  } |

| **Compliant Code** |
| --- |
| Ensure the index is within the bounds of the array before accessing. |
| T get\_value(size\_t index) {  int arr[5] = { 0 };  if (index > arr.size()) return –1;  T value = arr[index];  return value;  } |

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

| **Principles(s):** The rule in SEI CERT that is associated with this standard is CTR50-CPP. This rule states that you must guarantee that iterators and indices are within the valid. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.3p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TO  LANG.MEM.TU  LANG.MEM.TBA  LANG.STRUCT.PBB  LANG.STRUCT.PPE  LANG.STRUCT.PARITH | Buffer overrun, underrun  Type overrun, underrun  Tainted buffer access  Pointer before beginning of object  Pointer past end of object  Pointer arithmetic |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Mitigate buffer overflow risk by validating the size of strings before writing to their memory. |

| **Noncompliant Code** |
| --- |
| strcpy can write arbitrary values to memory if the source string is too long to fit into the destination string. |
| char dest[10];  strcpy(dest, src); |

| **Compliant Code** |
| --- |
| strncpy only writes the first n characters of the source string. |
| char dest[10];  strncpy(dest, source, sizeof(dest) - 1);  dest[sizeof(dest) - 1] = ‘\0’; |

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

| **Principles(s):** STR07-C states that you must use the bounds-checking interfaces for string manipulation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PC-lint Plus | 1.4 | 586 | Fully supported checks for violations of this rule |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Prevents SQL injection by separating SQL commands from data. |

| **Noncompliant Code** |
| --- |
| Directly concatenating user input into queries allows malicious users to run arbitrary SQL commands. |
| string query = “SELECT \* FROM users WHERE username = ‘” + user\_input + “’”; |

| **Compliant Code** |
| --- |
| Utilize parameterized queries to sanitize user inputs before running SQL commands. |
| sql::PreparedStatement\* stmt = conn->prepareStatement(“SELECT \* FROM users WHERE id=?”);  stmt->setString(1, user\_input); |
| See <https://dev.mysql.com/doc/connector-cpp/1.1/en/connector-cpp-examples-prepared-statements.html> for documentation on prepared statements with MySQL in C++. |

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

| **Principles(s):** ENV03-C states that you must sanitize the environment when invoking external programs. For example, sanitize user input before running SQL queries. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.4 | C5017 | A static analysis tool for C and C++ preferred in tightly regulated and safety-critical industries. |
| LDRA tool suite | 9.7.1 | 588 S | Static and dynamic software analysis tool. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Use smart pointers to prevent common memory errors such as memory leaks and dangling pointers. |

| **Noncompliant Code** |
| --- |
| Manually allocating memory without deallocating causes a memory leak. |
| int\* ptr = new int[100];  // no delete |

| **Compliant Code** |
| --- |
| Using smart pointers prevents memory leaks and dangling pointers. |
| int[] arr = new int[100];  std::unique\_ptr<int[]> = std::make\_unique<int[]>(arr); |

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

| **Principles(s):** MEM51-CPP - Properly deallocate dynamically allocated resources. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | invalid\_dynamic\_memory\_allocation  dangling\_pointer\_use | Static analyzer for C and C++ build for embedded software. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-CPP-006 | Assertions should be used only for debugging purposes. See https://cwe.mitre.org/data/definitions/617.html. |

| **Noncompliant Code** |
| --- |
| An attacker can trigger an assertion to fail if assertions are left in production builds, causing programs to fail and risking the availability of the application. |
| assert(user\_input < 10);  // user enters 11, causing program crash |

| **Compliant Code** |
| --- |
| Use runtime checks instead of assertions in production builds. |
| if (user\_input >= 10) {  // handle error gracefully  } |

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

| **Principles(s):** ERR50-CPP – Do not abruptly terminate the program. This could be exploited for denial-of-service attacks or leave the executing environment vulnerable due to not calling exit handlers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.3p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort & exit |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handling exceptions gracefully prevents exposing sensitive data. It also introduces an attack vector for malicious actors to reduce the availability of an application. |

| **Noncompliant Code** |
| --- |
| Logging sensitive data when exceptions are thrown leaks that data to the user. |
| try {  authenticate(credentials);  } catch (...) {  cerr << “Invalid credentials: “ << credentials << std::endl;  } |

| **Compliant Code** |
| --- |
| Handle exceptions gracefully and do not give unnecessary information about the error. |
| try {  authenticate(credentials);  } catch (...) {  cerr << “Authentication failed” << std::endl;  } |

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

| **Principles(s):** ERR57-CPP - Do not leak resources when handling exceptions. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.3p0 | ALLOC.LEAK | Leak |
| Polyspace Bug Finder | R2024a | CERT C++: ERR57-CPP | Checks for resource leak caused by exception |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Environment Variables | STD-CPP-008 | Use environment variables for API keys and secrets, database users and passwords, and other sensitive information required for development. Never push a .env file to source control. |

| **Noncompliant Code** |
| --- |
| Avoid hard-coding sensitive data into source code. |
| driver = sql::mysql::get\_mysql\_driver\_instance();  con = driver->connect("tcp://127.0.0.1:3306", "user", "password"); |

| **Compliant Code** |
| --- |
| Utilize environment variables to prevent leaking credentials to source control. |
| driver = sql::mysql::get\_mysql\_driver\_instance();  con = driver->connect(std::getenv(“DB\_URL”), std::getenv(“DB\_USER”), std::getenv(“DB\_PASSWORD”)); |

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

| **Principles(s):** MSC41-C. Never hard code sensitive information. This can get leaked in production code and exposed to the end user. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Code | 8.3p0 | HARDCODED.AUTH  HARDCODED.DNS  HARDCODED.KEY  HARDCODED.SALT  HARDCODED.SEED | Detects hardcoded authorization information, DNS name, cryptography keys, salts, and seeds. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Update Dependencies | STD-009-CPP | Outdated dependencies commonly contain patched vulnerabilities. |

| **Noncompliant Code** |
| --- |
| Using a deprecated library or function with a known vulnerability is an easy way for an attacker to hack into a system by simply looking up vulnerabilities for the libraries an application is using. |
| old\_lib\_version::some\_deprecated\_func(); // WARNING: CVE-XXX-XXXX |

| **Compliant Code** |
| --- |
| Install the latest versions of the libraries and schedule regular system updates to maintain security. |
| stable\_lib\_version::some\_func(); // Safe, patched known vulnerabilities |

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

| **Principles(s):** MSC24-C - Do not use deprecated or obsolescent functions. Dependencies are often patched to fix known security vulnerabilities. Using old versions of libraries allows hackers to exploit known vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MSC24 | Fully implemented checks for use of deprecated |
| ECLAIR | 1.2 | CC2.MSC34 | Fully implemented checks for use of deprecated functions |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Cryptographic Algorithms | STD-CPP-010 | Use stable and secure cryptographic algorithms as outdated ones may have vulnerabilities. As computers have gotten more powerful, cryptographic algorithms need to be more and more secure. |

| **Noncompliant Code** |
| --- |
| Using weak cryptographic algorithms leaves vulnerabilties in a system. |
| void hash\_password(const std::string& pw) {  std::string hashed = MD5(pw); // MD5 is weak to brute force attacks  } |

| **Compliant Code** |
| --- |
| Using secure cryptographic algorithms keeps sensitive user data protected. |
| void hash\_password(const std::string& pw) {  std::string hashed = SHA256(pw);  } |

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

| **Principles(s):** MSC18-C – Be careful while handling sensitive data, such as passwords, in program code. Do not implement your own encryption functions. Instead, use crypto libraries that have proven to be secure. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2024a | CERT C: Rec. MSC18-C | Checks for unsafe standard encryption functions and constant or predictable cipher keys. |

### 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.

Throughout the DevSecOps pipeline, automation can be used to reduce risk of errors, increase efficiency, and reduce the cost of maintaining a security process. By introducing automation, the workload of the DevSecOps team is also lowered. In the planning stage, automation tools can be utilized to scan vulnerability databases and look for threats related to the system being developed. This helps the team remain compliant with regulations surrounding the field and to security best practices. In the build stage, static analysis tools can be used to check for vulnerabilities in the codebase as the code is written. These tools can be built into the developer’s IDE to provide them with real-time information about the security of the code they are writing so that they can correct any potential vulnerabilities before they finish implementing a function. It is helpful to have a mandated IDE and static analysis tool for all developers to use to ensure that the team is following the same principles. With that being said, a range of tools can be utilized to cover the widest range of vulnerabilities, as some tools may not check for certain vulnerabilities, while others do.

In the verify and test stage, automated unit tests should be written and ran regularly to check for threats identified by the thread modeling process. These unit tests should attempt to perform unsafe operations using the code written and ensure that each case is handled safely and in compliance with the security standard agreed upon by the team, such as SEI CERT. In production, implementing a robust logging, detection, and response protocol ensures that the team is ready and knows how to react in the event of a potential security breach. The team should all be trained and know exactly what to do to prevent chaos from ensuing. Automated systems can send warnings to team members in the event of a detected anomaly so that it can be assessed and taken

### 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 | High | P9 | L2 |
| STD-003-CPP | High | Probable | Medium | P12 | L1 |
| STD-004-CPP | High | Likely | High | P9 | L2 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Probable | Medium | P4 | L3 |
| STD-007-CPP | Low | Probable | High | P2 | L3 |
| STD-008-CPP | High | Probable | Medium | P12 | L1 |
| STD-009-CPP | High | Probable | Medium | P12 | L1 |
| STD-010-CPP | Medium | Probable | Medium | P8 | 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 | Secure data that is stored in non-volatile memory such as hard drives. Maintains confidentiality and integrity of data to prevent unauthorized access and tampering. In organizations operating in regulated fields, ensures compliance with data laws. |
| Encryption in flight | Secures data that is being transmitted over a network. Attackers can intercept network transmissions and read and alter the data being sent before it gets to the receiver, known as a man-in-the-middle attack. |
| Encryption in use | Secure data that is currently in volatile memory. When the data is being used for computations, attackers could gain access to the process that holds the data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Ensures that an entity is who it claims to be. Authentication verifies that an entity that claims to be authorized actually has legitimate authorization. Weak authentication is vulnerable to breaches from an entity that appears to be authorized but isn’t. When a user is logged in and makes a request, then send their authorization key which is authenticated. |
| Authorization | Grants access to a system by verifying an entity’s credentials and assigning them some. Entities should only be granted the least amount of privilege possible to perform their task. More robust authorization should be required to perform more sensitive tasks. This protects changes to databases as it prevents changing a table or column that you shouldn’t be able to. When adding new users, they are granted only certain authorizations depending on their role. |
| Accounting | Logs requests to the system, keeping track of information about the entity and its behavior. Accounting enables the analysis of activity during a security incident to prevent future attacks. It is also useful for preventing attacks as they happen when they are detected. Accessing and changing sensitive files should be monitored closely and incidents sent to the appropriate team. |

**\***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 first standard ensures the use of the correct data types to prevent overflow and underflow. This follows the security principle of having fail-safe defaults. The second standard prevents out of bounds indexing by checking to make sure the index is within a valid range. This standard maps to defense in depth by adding an additional layer of protection and secure by design. The third standard validates string sizes to prevent buffer overflow attacks. This validation reduces the attack surface and maps to the principle of minimizing the attack surface. The fourth standard prevents SQL injections and maps to the principle to validate inputs. The fifth standard enforces the use of smart pointers to prevent memory leaks and dangling pointers, and follows the security principle of secure by design. The sixth standard prevents the use of assertions in production code, and adheres to the principle of failing securely. The seventh standard is to handle exceptions securely and maps to the principle of failing securely. The eighth standard states that environment variables should be used to prevent leaking secrets in source code, and maps to the principle of secure storage and minimizing the attack surface. The ninth standard states that dependencies should be updated and maps to the principle of secure by design. The tenth standard is to use secure cryptographic algorithms only and maps to the standard of defense in depth and sec

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 | 01/26/2025 | Create 10 secure coding standards | Bobby Rust |  |
| 1.2 | 02/16/2025 | Finish standards, triple-A and encryption | Bobby Rust |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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