**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 4](#_Toc52464058)

[Coding Standard 1 5](#_Toc52464059)

[Coding Standard 2 7](#_Toc52464060)

[Coding Standard 3 9](#_Toc52464061)

[Coding Standard 4 11](#_Toc52464062)

[Coding Standard 5 13](#_Toc52464063)

[Coding Standard 6 15](#_Toc52464064)

[Coding Standard 7 17](#_Toc52464065)

[Coding Standard 8 19](#_Toc52464066)

[Coding Standard 9 21](#_Toc52464067)

[Coding Standard 10 23](#_Toc52464068)

[Defense-in-Depth Illustration 25](#_Toc52464069)

[Project One 25](#_Toc52464070)

[1. Revise the C/C++ Standards 25](#_Toc52464071)

[2. Risk Assessment 25](#_Toc52464072)

[3. Automated Detection 25](#_Toc52464073)

[4. Automation 25](#_Toc52464074)

[5. Summary of Risk Assessments 26](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 26](#_Toc52464076)

[7. Map the Principles 28](#_Toc52464077)

[Audit Controls and Management 29](#_Toc52464078)

[Enforcement 29](#_Toc52464079)

[Exceptions Process 29](#_Toc52464080)

[Distribution 30](#_Toc52464081)

[Policy Change Control 30](#_Toc52464082)

[Policy Version History 30](#_Toc52464083)

[Appendix A Lookups 30](#_Toc52464084)

[Approved C/C++ Language Acronyms 30](#_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 | The principle of validating input data shows how important it is for all data coming into a system, whether from users, external systems, or internal parts, to go through a rigorous validation process before being used. This type of validation safeguards against potential security threats. For example, SQL injection and cross-site scripting by ensuring that the data conforms to the expected format and stays within acceptable limits. Input validation is a defense against attacks from, and it is also a key part of keeping data in the system accurate and consistent. Depending on the specific use case, this process could include checks for data type, range, format, and other properties. This would make the software more secure and reliable. |
| 1. Heed Compiler Warnings | Compiler warnings are often seen as optional and not very important, you shouldn't ignore them. These warnings could be signs that there are holes in the code, which might not be a threat right now but could cause security problems in the future. Taking care of these warnings makes the software more secure against security threats and makes the code more efficient and easier to maintain. So, developers should try to eliminate all warnings in their codebase and treat them as possible errors that must be looked at. This proactive software quality and security approach can make the codebase cleaner and stronger. |
| 1. Architect and Design for Security Policies | The security policies of an organization should be thought about from the beginning of the software development process. The architecture and design of the software should be made in line with these policies to make sure that security is not just an afterthought but is built into the process of making the software. By thinking about security from the beginning, potential holes or weak spots can be found and fixed early on, making them easier to fix. The organization's security policies, such as data protection, user authentication, and access controls, should be supported by a strong and reliable software architecture. |
| 1. Keep It Simple | The principle of keeping code as simple as possible shows that complexity often leads to security holes. Keeping the codebase as simple and clear as possible helps people understand it better, which makes it easier to find security problems and fix them. Complex codebases can hide security holes and make it hard for developers to see what their changes might mean. Putting an emphasis on simplicity in code design doesn't mean less functionality or sophistication. Instead, it encourages code that is clear, concise, and straightforward, which makes it easier to maintain and less likely to have mistakes during implementation. |
| 1. Default Deny | The "default deny" principle says that all system requests should be turned down by default, unless they are given permission. This method reduces the chance of unauthorized access or actions because it treats every process as if it could be a threat. This principle, which applies to all parts of the system, encourages proactive security measures, reduces the system's attack surface, and protects against both internal and external threats. Even though this method may take more work to set up, the increased security it provides makes it worth it. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege says that a user or process should only have as many permissions as they need to do their job. This basic security principle helps to limit the damage that could be done if a user account or process is broken into. By stopping processes from using resources they don't need, security problems can be kept to a minimum. This method needs careful planning, a thorough understanding of each user's roles and responsibilities, and careful implementation, but it pays off in terms of security in a big way. |
| 1. Sanitize Data Sent to Other Systems | Before sending data to another system, it is important to clean the data. During this process, any potentially sensitive or unnecessary information is taken out. This protects the privacy of user data and the integrity of the system. During the process of sanitization, it may be necessary to escape special characters, get rid of null bytes, or encode binary data. It is very important to know how the data will be used in the system that receives it, as this affects how the data is cleaned. This principle reduces the risk of data leaks and helps people follow the rules about data protection. |
| 1. Practice Defense in Depth | Using the "defense in depth" principle means putting in place multiple layers of security controls in an IT system. This idea comes from military strategies. It involves putting several layers of defense into a system to keep it safe even if one layer fails. These layers can have network firewalls, intrusion detection systems, encryption in transit and at rest, regular software updates, and strict access controls. Defense in depth creates backups and stops a single point of failure from putting the security of the whole system at risk. |
| 1. Use Effective Quality Assurance Techniques | Before software is released, it is important to find any potential security problems by using good quality assurance (QA) techniques. Techniques for quality assurance like code reviews, unit tests, and integration tests help find security holes early on. QA isn't just about finding bugs, though. It's also about making sure that the system works as expected in different situations and handles errors well. All parts of the system should be covered by a thorough test plan that is constantly updated. Regular security audits and penetration testing are also important parts of the QA process. The main goal of quality assurance is to make it less likely that security flaws will get into the production environment. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard gives developers a set of rules to follow when writing code that minimizes security risks. These standards may have suggestions for how to handle user input, manage memory, handle errors, and more. Following a secure coding standard can make software more consistent, reduce the chance of mistakes, and make the software more secure. Secure coding standards are made by industry experts and based on best practices and lessons learned from past security incidents. They are updated often to reflect changes in technology and threats. Important parts of this principle are also training and ongoing education for developers in secure 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] | Strong typedefs in C++ improve type safety and create clear, semantically correct code. |

| **Noncompliant Code** |
| --- |
| In the following block, a simple typedef is used which can lead to confusion and errors, as it doesn't differentiate between different uses of the same base type. |
| int a;  std::cout << a; // undefined behavior |

| **Compliant Code** |
| --- |
| In the following block of code, a strong typedef (using a struct) is used to provide a clear differentiation between different uses of the same base type. |
| struct Length {  explicit Length(int l) : val(l) {}  int val;  };  struct Weight {  explicit Weight(int w) : val(w) {}  int val;  };  Length l(5);  Weight w = l; // Compiler error |

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

| **Principles(s):** Strong Type Safety is a principle that enforces stricter type checking, ensuring that different types, even if they share the same underlying datatype, are not treated as interchangeable. This principle directly maps to the "Use of Strong Typedefs" standard. By creating a new, distinct type with a strong typedef, we are effectively enforcing stronger type safety in our code, reducing the possibility of type-related errors and improving the overall quality and reliability of the code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Moderate | Low | Low | Hight | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.6 | CPPCheck (typedef) | CPPCheck is a static analysis tool for C/C++ code |
| Clang-Tidy | 13.0.6 | Clang-Tidy (modernize-use-using) | Clang-Tidy is a clang-based C++ linter tool, |
| SonarQube | 10 | SonarQube (Simplify Type Names) | SonarQube is an open-source platform for continuous inspection of code quality |
| ReSharper C++ | 2023 | ReSharper C++ (Type Safety) | ReSharper C++ is a Visual Studio extension for C++ development |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Prefer Enumerated Types over Magic Numbers |

| **Noncompliant Code** |
| --- |
| It can be challenging to understand and handle the code when it employs arbitrary numerical values to indicate different colors. |
| int color = 3; // What does 3 mean? |

| **Compliant Code** |
| --- |
| The code uses an enumerated type to show various colors, which improves clarity and comprehension. |
| enum Color { RED, GREEN, BLUE, YELLOW };  Color color = YELLOW; // It's clear that the color is yellow |

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

| **Principles(s):** An essential principle in coding is "self-documenting code." It's crucial to write code that is easy to understand without additional comments. Enumerated types should be used instead of "magic numbers" to achieve this goal. By doing so, the meaning of each value is clear, making the code more self-explanatory. This approach enhances the readability, comprehension, and maintainability of the code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.6 | CPPCheck (avoid magic numbers) | CPPCheck is a static analysis tool for C/C++ code |
| Clang-Tidy | 13.0.6 | Clang-Tidy (readability-magic-numbers), | Clang-Tidy is a clang-based C++ linter tool, |
| SonarQube | 10 | SonarQube (Magic numbers should not be used). | SonarQube is an open-source platform for continuous inspection of code quality |
| ReSharper C++ | 2023 | ReSharper C++ (Replace magic number with named constant) | ReSharper C++ is a Visual Studio extension for C++ development |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | String Length Limit |

| **Noncompliant Code** |
| --- |
| The code below does not impose a limit on input string length. This lack of constraint could significantly impact memory and performance if a string exceeds a certain length. |
| def process\_string(input\_string):  # Process the input string without checking its length  ... |

| **Compliant Code** |
| --- |
| This code sets a maximum length for the input string, preventing it from going over that limit. |
| MAX\_STRING\_LENGTH = 100  def process\_string(input\_string):  if len(input\_string) > MAX\_STRING\_LENGTH:  raise ValueError("Input string exceeds the maximum allowed length.")    # Process the input string  ... |

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

| **Principles(s):** To ensure that strings are correct, we enforce a length limit on input strings through our coding standard. This approach aligns with the principle of robustness by preventing any potential issues that could arise from handling excessively long strings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.6 | CPPCheck (buffer.overrun) | CPPCheck is a static analysis tool for C/C++ code |
| Clang-Tidy | 13.0.6 | Clang-Tidy (bugprone-string-constructor) | Clang-Tidy is a clang-based C++ linter tool, |
| SonarQube | 10 | SonarQube (S3518). | SonarQube is an open-source platform for continuous inspection of code quality |
| ReSharper C++ | 2023 | ReSharper C++ (Buffer Overflow) | ReSharper C++ is a Visual Studio extension for C++ development |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | To prevent security vulnerabilities, a standard called Data Security recommends the use of parameterized queries or prepared statements. |

| **Noncompliant Code** |
| --- |
| When inserting user input into a SQL query as it may result in SQL Injection if the input contains SQL code. |
| string query = "SELECT \* FROM users WHERE username = '" + username + "' AND password = '" + password + "'"; |

| **Compliant Code** |
| --- |
| To prevent SQL Injection, it is best to use a Parameterized query. This ensures that the user input is treated as a literal string, which means that it cannot change the structure of the SQL query. |
| string query = "SELECT \* FROM users WHERE username = ? AND password = ?"; |

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

| **Principles(s):** The Principle of Least Privilege states that users should only have access to the necessary levels of information to complete their tasks. Specifically, in regards to this standard, user input should not impact the structure of the SQL query. To prevent this, parameterized queries or prepared statements can be utilized to ensure that user input cannot alter the SQL query, effectively following the Principle of Least Privilege. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | Medium | Medium | Critical | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.6 | CPPCheck (SqlInjection) | CPPCheck is a static analysis tool for C/C++ code |
| Clang-Tidy | 13.0.6 | Clang-Tidy (cppcoreguidelines-pro-type-vararg) | Clang-Tidy is a clang-based C++ linter tool, |
| SonarQube | 10 | SonarQube (S3649) | SonarQube is an open-source platform for continuous inspection of code quality |
| ReSharper C++ | 2023 | ReSharper C++ (SqlInjection) | ReSharper C++ is a Visual Studio extension for C++ development |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | To prevent a vulnerability called Buffer Overflow, it's important to follow the standard of not writing beyond the capacity of buffers. |

| **Noncompliant Code** |
| --- |
| In the code below, if the input string is too long, it may cause a buffer overflow error. |
| char buffer[10];  strcpy(buffer, userInput); // No check on the size of userInput |

| **Compliant Code** |
| --- |
| To prevent buffer overflow, a check has been added to ensure that the input string does not exceed the size of the buffer in the following block. |
| char buffer[10];  if (strlen(userInput) < sizeof(buffer)) {  strcpy(buffer, userInput);  } else {  // Handle the error  } |

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

| **Principles(s):** According to the Principle of Defense in Depth, a system should have several layers of security. This ensures that if one layer fails, others are still in place to prevent a security breach. In this context, it means adding checks in the code to prevent writing beyond the buffer's capacity, even if the input size is limited elsewhere. By doing so, we are adding an extra layer of security, which aligns with the Principle of Defense in Depth. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | Medium | High | Critical | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.6 | CPPCheck (buffer.overrun) | CPPCheck is a static analysis tool for C/C++ code |
| Clang-Tidy | 13.0.6 | Clang-Tidy (bugprone-array-index) | Clang-Tidy is a clang-based C++ linter tool, |
| SonarQube | 10 | SonarQube (S3518) | SonarQube is an open-source platform for continuous inspection of code quality |
| ReSharper C++ | 2023 | ReSharper C++ (Buffer Overflow) | ReSharper C++ is a Visual Studio extension for C++ development |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | To validate program invariants and identify logic errors early in the development process, it is recommended to use assertions. This standard emphasizes the importance of utilizing assertions. |

| **Noncompliant Code** |
| --- |
| A function expects the input parameter to be valid. If an invalid parameter is used, the function may not work as intended. |
| void squareRoot(double x) {  return std::sqrt(x); // Undefined behavior if x is negative  } |

| **Compliant Code** |
| --- |
| In order to validate the input parameter, an assertion is utilized. If an invalid parameter is passed, the program will terminate, making it simpler to identify and resolve the problem. |
| void squareRoot(double x) {  assert(x >= 0); // Program terminates if x is negative  return std::sqrt(x);  } |

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

| **Principles(s):** The Fail-Fast Principle is a guideline for software development which suggests that a system should quickly notify the user if there is a potential for failure. It is more advisable to experience failure during development or testing than in a production environment. Assertions are used to enable the program to fail immediately upon detecting an unexpected condition. This helps identify and correct errors early on, reducing the likelihood of encountering these problems during production. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.6 | CPPCheck (missingAssertion) | CPPCheck is a static analysis tool for C/C++ code |
| Clang-Tidy | 13.0.6 | Clang-Tidy (bugprone-assert-side-effect) | Clang-Tidy is a clang-based C++ linter tool, |
| SonarQube | 10 | SonarQube (S5803) | SonarQube is an open-source platform for continuous inspection of code quality |
| ReSharper C++ | 2023 | ReSharper C++ (Assertion) | ReSharper C++ is a Visual Studio extension for C++ development |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | The Proper Use of Exceptions standard encourages using exceptions appropriately to handle unusual situations, preventing system crashes or undefined behavior. |

| **Noncompliant Code** |
| --- |
| If an exception is not caught, it can result in the termination of the program. |
| void mightThrow() {  throw std::runtime\_error("An error occurred");  } // No catch block to handle the exception |

| **Compliant Code** |
| --- |
| If an exception occurs, it will be caught and handled, so the program won't terminate. |
| void mightThrow() {  try {  throw std::runtime\_error("An error occurred");  } catch (const std::exception& e) {  std::cerr << e.what(); // Handle the exception  }  } |

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

| **Principles(s):** The Robustness Principle guides software development by suggesting that programs should be able to handle unexpected or exceptional conditions without crashing. This is achieved by catching and handling exceptions in a controlled manner, which makes programs more stable and less prone to crashing. By following this principle, we can ensure that our software is more resilient and reliable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.6 | CPPCheck (exceptionSafety) | CPPCheck is a static analysis tool for C/C++ code |
| Clang-Tidy | 13.0.6 | Clang-Tidy (cppcoreguidelines-exceptions) | Clang-Tidy is a clang-based C++ linter tool, |
| SonarQube | 10 | SonarQube (S5197) | SonarQube is an open-source platform for continuous inspection of code quality |
| ReSharper C++ | 2023 | ReSharper C++ (Exception) | ReSharper C++ is a Visual Studio extension for C++ development |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Properly Managing Sensitive Information | [STD-008-CPP] | Secure Handling of Sensitive Information |

| **Noncompliant Code** |
| --- |
| The user passwords are stored in the database as plain text. If an attacker were to gain access to this information, they would have immediate access to the users' passwords. |
| def create\_user(username, password):  query = f"INSERT INTO users (username, password) VALUES ('{username}', '{password}')"  return execute\_query(query) |

| **Compliant Code** |
| --- |
| To enhance security, we store a hash of the password using a cryptographic hash function instead of the password itself. This way, if there is a data breach, the attacker will only have access to the hashes and not the actual passwords. |
| import hashlib  def create\_user(username, password):  password\_hash = hashlib.sha256(password.encode()).hexdigest()  query = "INSERT INTO users (username, password) VALUES (?, ?)"  return execute\_query(query, (username, password\_hash)) |

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

| **Principles(s):** To improve security, it's important to minimize the attack surface area by limiting the amount of code and features that could be vulnerable to exploitation by attackers. One way to do this is by hashing sensitive data like passwords before storing them in the database, which can reduce potential damage in the event of a breach. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | High | High | Critical | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.6 | CPPCheck (sensitiveInfo) | CPPCheck is a static analysis tool for C/C++ code |
| Clang-Tidy | 13.0.6 | Clang-Tidy (cppcoreguidelines-sensitive-data) | Clang-Tidy is a clang-based C++ linter tool, |
| SonarQube | 10 | SonarQube (S5643) | SonarQube is an open-source platform for continuous inspection of code quality |
| ReSharper C++ | 2023 | ReSharper C++ (Sensitive Information) | ReSharper C++ is a Visual Studio extension for C++ development |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Cross-Site Scripting Avoidance | [STD-009-JS] | Prevention of Cross-Site Scripting Attacks |

| **Noncompliant Code** |
| --- |
| When the user input is displayed on the web page, it is not sanitized or encoded. This can create a security weakness known as Cross-Site Scripting, which allows hackers to inject harmful scripts into the pages viewed by other users. |
| <script>  var userInput = get\_user\_input(); // function to get user input  document.getElementById("demo").innerHTML = userInput;  </script> |

| **Compliant Code** |
| --- |
| Before being outputted, all user input is thoroughly sanitized to prevent it from being interpreted as code. |
| <script>  var userInput = get\_user\_input(); // function to get user input  document.getElementById("demo").textContent = userInput;  </script> |

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

| **Principles(s):** Principle of Least Privilege - This standard corresponds with the principle of least privilege. It advocates that code should not have more privilege than necessary. In this case, the user input should not have the privilege to be executed as part of the code, thereby preventing Cross-Site Scripting attacks. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | High | High | Critical | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ESLint | 8.0.0 | ESLint (no-unescaped-entities) | ESLint is a tool for identifying and reporting on patterns found in ECMAScript/JavaScript code |
| SonarJS | 7.3 | SonarJS (S5131) | SonarJS is a static code analyzer for JavaScript, |
| Snyk | 1.704.0 | Snyk (SNYK-JS-XXS) | Snyk is a developer-first security tool that finds and fixes vulnerabilities in your code |
| OWASP Zap | 2.12.0 | OWASP Zap (40012) | OWASP Zap is a full-featured, open-source web application security scanner. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Random Number Generators | [STD-010-CPP] | Safe Usage of Random Number Generators |

| **Noncompliant Code** |
| --- |
| This particular function is not recommended for creating cryptographic keys or other security-related numbers as it does not offer enough randomness. |
| var secretToken = Math.random(); |

| **Compliant Code** |
| --- |
| To create a secret token, we utilize a secure random number generator such as crypto.randomBytes(). |
| var crypto = require('crypto');  var secretToken = crypto.randomBytes(64).toString('hex'); |

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

| **Principles(s):** The Defense in Depth standard recommends having several layers of security measures to defend against potential threats. One way to add an extra layer of security is by using a secure random number generator. Even if other security measures fail, the unpredictability of the secret token generated by a secure RNG can prevent attackers from guessing the token. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.6 | CPPCheck (insecureRandomness) | CPPCheck is a static analysis tool for C/C++ code |
| Clang-Tidy | 13.0.6 | Clang-Tidy (cert-msc50-cpp) | Clang-Tidy is a clang-based C++ linter tool, |
| SonarQube | 10 | SonarQube (S2278) | SonarQube is an open-source platform for continuous inspection of code quality |
| ReSharper C++ | 2023 | ReSharper C++ (Weak RNG), | ReSharper C++ is a Visual Studio extension for C++ development |

### 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.

[Insert your written explanations here.]

### 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 | Moderate | Low | Low | Hight | 2 |
| STD-002-CPP | High | Medium | Medium | High | 2 |
| STD-003-CPP | High | Medium | Medium | High | 2 |
| STD-004-CPP | Critical | Medium | Medium | Critical | 1 |
| STD-005-CPP | Critical | Medium | High | Critical | 1 |
| STD-006-CPP | Medium | High | Medium | Medium | 3 |
| STD-007-CPP | High | Medium | Medium | High | 2 |
| STD-008-CPP | Critical | High | High | Critical | 1 |
| STD-009-CPP | High | Low | Low | High | 1 |
| STD-001-JS | Critical | High | High | Critical | 1 |

### 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 in rest | The technique called "data encryption at rest" is used to protect stored data by encoding it while it's on a disk. This ensures that the data can only be accessed with the correct decryption key. The purpose of this approach is to safeguard data on storage devices such as hard drives, SSDs, servers, and data centers. It's a necessary measure to prevent unauthorized access and data breaches, particularly in the case of physical theft or attacks. This policy applies to sensitive data that is stored and not in use. |
| Encryption at flight | When data is being moved between networks or within networks, it is important to protect it from unauthorized interception. This is where encryption in flight, or encryption in transit, comes in. The process involves encrypting data while it is in transit to ensure its security. Commonly used protocols such as HTTPS, SSL, or TLS are employed to achieve this. It is important to apply this policy to all data in transit, especially over untrusted networks, to prevent "man-in-the-middle" attacks. |
| Encryption in use | When encryption is in use, it means that data is being encrypted while it is being processed. This type of encryption is more challenging to set up compared to encryption at rest or in transit because the data must be decrypted first before processing. As more businesses rely on cloud services and outsourced processing, this concept is becoming more crucial. The policy is relevant for scenarios where sensitive data is processed outside of trusted boundaries, like third-party services or public cloud services. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication involves confirming the identity of a person, system, or process. This is achieved by verifying the accuracy of their credentials, such as usernames, passwords, tokens, or biometric data. It's an essential part of maintaining secure access control in a system and is the first pillar of the Triple-A framework. This policy applies to all logins to ensure that each user is truly who they say they are, preventing unauthorized access or potential breaches. |
| Authorization | Once a user has been authenticated, the next step is authorization, which involves deciding whether to grant or deny access to specific resources or actions based on the user's profile or role. In an RBAC system, users are assigned roles, and permissions are assigned to those roles to simplify access management. The policy applies to all system components, databases, files, and services, regulating user activities and preventing unauthorized actions, even in the event of a system breach. |
| Accounting | Tracking user activities and operations is known as accounting or auditing. This involves maintaining a record of the user's actions, such as system entry and exit times, executed commands, accessed files, and changes made. Accounting enables the review and analysis of resource utilization, providing traceability. This policy applies to all user interactions with the system to monitor behavior, detect anomalies, and support security investigations when required. |

**\***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  
    
  When users log in, we check their credentials against our stored database to ensure a match. Before any changes are made to the database, we make sure users have appropriate permissions. When new users are added, we authenticate their identity and authorize their level of access. User access is determined by their role and responsibilities. We also keep track of which files are accessed by users and when through our accounting system.

### 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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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 |