**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 | This principle emphasizes the importance of verifying and validating all incoming data to ensure that it meets the expected format and criteria. By doing so, we can prevent malicious data from entering our systems, reducing the risk of vulnerabilities and security breaches. |
| 1. Heed Compiler Warnings | Paying attention to compiler warnings is crucial. These warnings often highlight potential issues or vulnerabilities in the code. Addressing these warnings helps in identifying and rectifying security weaknesses before they can be exploited by attackers. |
| 1. Architect and Design for Security Policies | When designing systems and applications, security should be an integral part of the architecture. This principle encourages us to consider security requirements and policies from the outset, ensuring that security is not an afterthought but a fundamental aspect of our design. |
| 1. Keep It Simple | Simplicity is a key principle in security. Complex systems often introduce more vulnerabilities and are harder to secure. By keeping systems and code simple and straightforward, we can reduce the attack surface and make it easier to identify and address security issues. |
| 1. Default Deny | This principle suggests that by default, access should be denied to all resources or systems. Only when access is explicitly allowed, based on predefined rules and policies, should it be granted. This approach minimizes the risk of unauthorized access and data breaches. |
| 1. Adhere to the Principle of Least Privilege | Users and systems should only be given the minimum level of access or permissions necessary to perform their tasks. This limits potential damage in case of a security breach and reduces the chances of unauthorized actions. |
| 1. Sanitize Data Sent to Other Systems | Before sending data to external systems or resources, it should be thoroughly sanitized and validated. This prevents the transmission of malicious or unintended data that could compromise the integrity of the recipient system. |
| 1. Practice Defense in Depth | This principle advocates for a layered approach to security. Instead of relying on a single security measure, we should implement multiple layers of defense, making it more difficult for attackers to penetrate the system. This includes firewalls, intrusion detection systems, encryption, and more. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance processes should include security testing and validation. By rigorously testing software and systems for security vulnerabilities, we can identify and rectify issues before they are exploited by attackers. |
| 1. Adopt a Secure Coding Standard | We should establish and follow secure coding standards and best practices throughout the software development lifecycle. This ensures that code is written with security in mind, reducing the likelihood of vulnerabilities being introduced during development. |

### 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 | This standard ensures that variables and data structures are declared with appropriate data types to avoid unintended type conversions and data corruption. |

| **Noncompliant Code** |
| --- |
| In this example, an integer is assigned to a variable declared as a char. This can lead to truncation and data loss. |
| char myChar = 65; // integer used in a character data type |

| **Compliant Code** |
| --- |
| To comply with the Data Type standard, use the correct data type for the intended value. In this case, we use an integer data type. |
| int myInt = 65; // integer used in an integer data type |

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

| **Principles(s):**  ***1. "Validate Input Data"*** - The principle is relevant to this standard as it ensures that data input is correctly typed and validated, reducing the risk of data corruption and type-related vulnerabilities.  ***3. “Architect and Design for Security Policies” -*** The principle is relevant to this standard as ensures that data security is being addressed when designing the system, ensuring proper data types are used throughout the system, reducing the risk of data corruption and type-related vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang-Tidy](https://clang.llvm.org/extra/clang-tidy/checks/list.html) | 18.0.0 | cppcoreguidelines-pro-type-vararg | This checker detects inappropriate use of variable arguments and suggests using safer alternatives. It helps identify type-related vulnerabilities in C++ code. |
| [CPPCheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 2.12 | typeMismatch | CppCheck's typeMismatch checker detects type mismatches, helping identify potential vulnerabilities related to incorrect data types. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | This standard ensures that variables are initialized with valid and appropriate values to prevent undefined behavior and unexpected results. |

| **Noncompliant Code** |
| --- |
| Variable is declared but not initialized, leading to unpredictable behavior. |
| int value; // variable not initialized |

| **Compliant Code** |
| --- |
| To comply with the Data Value standard, initialize variables with valid values. |
| int value = 65; // variable is initialized |

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

| **Principles(s):**  ***1. "Validate Input Data"*** - principle aligns with this standard by emphasizing the importance of initializing variables with valid values to prevent undefined behavior and unexpected results.  ***9. “Use Effective Quality Assurance Techniques”*** - The practice of initializing variables with valid values is an essential quality assurance technique that helps identify and rectify issues before they are exploited by attackers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CPPCheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | uninitializedMember | This checker detects uninitialized class members and helps identify potential issues related to data value initialization. |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | UNINIT | Coverity's UNINIT checker identifies uninitialized variables and helps ensure proper data value initialization. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | This standard ensures that string manipulations are performed safely to prevent buffer overflows and memory corruption. |

| **Noncompliant Code** |
| --- |
| Here there's no validation for the length of the string, which can lead to buffer overflows. |
| char destination[10];  strcpy(destination, "This is a long string");  // string will cause buffer overflow |

| **Compliant Code** |
| --- |
| To comply with the String Correctness standard, use safe string functions and ensure proper buffer size. Code below ensures only characters up to the size of the variable are captured. |
| char destination[20];  strncpy(destination, "This is a safe string", sizeof(destination));  // only correct buffer size is captured |

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

| **Principles(s):**  ***7. "Sanitize Data Sent to Other Systems" -*** principle is relevant here as it emphasizes the need to validate and sanitize data before sending it to external systems, reducing the risk of buffer overflows and memory corruption. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://clang.llvm.org/docs/analyzer/checkers.html#security-insecureapi-strcpy-c) | 3.9 | Security.insecureAPI.strcpy | This checker identifies unsafe use of the strcpy function, which can lead to buffer overflows. It helps enforce safe string handling. |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.0.7 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Coverity Static Analysis with these checkers helps detect and prevent issues related to string correctness, including buffer overflows, incorrect buffer sizes, overruns, and string size mismatches. This enhances the safety and security of string manipulations in the code |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | This standard ensures that user inputs are sanitized and properly escaped to prevent SQL injection attacks. |

| **Noncompliant Code** |
| --- |
| This code directly inserts user input into an SQL query without sanitizing it, leaving the application vulnerable to SQL injection. |
| #include <iostream>  #include <sqlite3.h>  std::string userInput = "'; DROP TABLE users --";  // Directly utilizing user input  std::string query = "SELECT \* FROM products WHERE name = '" + userInput + "'"; |

| **Compliant Code** |
| --- |
| To comply with the SQL Injection standard, use prepared statements or sanitize user inputs before using them in SQL queries. Code below utilizes prepared statements to prevent SQL injection |
| #include <iostream>  #include <sqlite3.h>  const char\* sql = "SELECT \* FROM USERS WHERE NAME = ?";  const char\* user\_input = "empty"; // Would hold user input  int result = sqlite3\_open(":memory:", &db);  // prepare statement  result = sqlite3\_prepare\_v2(db, sql, -1, &stmt, nullptr);  // bind user input to correct parameter in query  result = sqlite3\_bind\_text(stmt, 1, user\_input, -1, SQLITE\_STATIC);  // process query results  while (sqlite3\_step(stmt) == SQLITE\_ROW) {  std::cout << "User found: " << sqlite3\_column\_text(stmt, 1) <<  std::endl;  }  sqlite3\_finalize(stmt);  sqlite3\_close(db); |

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

| **Principles(s):**  ***1. “Validate Input Data” -*** Using prepared statements or sanitizing user inputs before using them in SQL queries is crucial for preventing SQL injection attacks, which are a form of malicious input data.  ***6. ”Default Deny” -*** By implementing strong SQL injection prevention, we are ensuring that hackers are unable to access the system and by sanitizing and utilizing prepared statements, we are linking to this principle  ***7. "Sanitize Data Sent to Other Systems" -*** principle aligns with this standard by emphasizing the importance of sanitizing user inputs to prevent SQL injection attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Fortify](https://vulncat.fortify.com/en/weakness?codelang=C%2FC%2B%2B&po=12) | 1.0 | HTTP\_Response\_Splitting SQL\_Injection\_\_Persistence SQL\_Injection | Fortify's SQL Injection rule detects code patterns that could lead to SQL injection vulnerabilities, helping secure database interactions. |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 7.5 | SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_ FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Coverity’s SQL Injection rule also detects code patterns that could lead to SQL injection vulnerabilities |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | This standard ensures that memory is allocated and freed correctly to prevent memory leaks and corruption. |

| **Noncompliant Code** |
| --- |
| In this example, memory is allocated but never freed, causing a memory leak. |
| void\* memoryBlock = malloc(100);  // Use the memoryBlock |

| **Compliant Code** |
| --- |
| To comply with the Memory Protection standard, always free memory after it's no longer needed. Code below utilizes the memory and then correctly frees it. |
| void\* memoryBlock = malloc(100);  // Use the memoryBlock  free(memoryBlock); |

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

| **Principles(s):**  ***2. “Heed Compiler Warnings” -*** Relates to this standard by ensuring that compiler warnings are addressed and corrected to prevent security issues before execution.  ***9. "Use Effective Quality Assurance Techniques" -*** principle relates to this standard by advocating for rigorous testing to identify and rectify memory-related vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Valgrind](https://valgrind.org/docs/manual/mc-manual.html) | [Insert text.] | Memcheck | Valgrind's Memcheck tool detects memory-related issues, including leaks and invalid memory access, helping ensure memory protection. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-MEM31 | The Axivion Bauhaus CertC-MEM31 can detect dynamically allocated resources that are not freed |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | This standard ensures that assertions are used to validate assumptions and detect unexpected conditions during development and testing. |

| **Noncompliant Code** |
| --- |
| In this code, there are no assertions to check for unexpected conditions. |
| int result = performCalculation();  // No assertions for result validity |

| **Compliant Code** |
| --- |
| To comply with the Assertions standard, use assertions to validate conditions. |
| int result = performCalculation();  // complete assertion  assert(result >= 0 && result <= 100); |

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

| **Principles(s):**  ***4. ”Keep it Simple” -*** By implementing assertions throughout the code, we are able to quickly and effectively test our code before implementation in a simple and clean way.  ***8. "Practice Defense in Depth" -***principle aligns with this standard by promoting a layered approach to security, including the use of assertions for runtime checks.  ***10. “Adopt a Secure Coding Standard” -*** By establishing a secure coding standard and utilizing it during the build phase, we can find and correct issues quickly in the development phase utilizing assertions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | ASSERT\_USAGE  ASSERT\_SIDE\_EFFECTS | Coverity's checkers identify potential issues with assertions, ensuring their effectiveness in runtime checks. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | This standard ensures that exceptions are used appropriately for error handling, improving code maintainability and reliability. |

| **Noncompliant Code** |
| --- |
| In this code, exceptions are not used for error handling, making it challenging to handle exceptional cases. |
| int divide(int a, int b) {  // no exceptions to handle errors    return a / b;  } |

| **Compliant Code** |
| --- |
| To comply with the Exceptions standard, use of exceptions for error handling should be utilized whenever possible. |
| int divide(int a, int b) {  // exception handling to handle error  if (b == 0) {  throw std::runtime\_error("Division by zero");  }    return a / b;  } |

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

| **Principles(s):**  ***8. "Practice Defense in Depth" -*** principle is relevant here as it advocates for multiple layers of security, including the appropriate use of exceptions for error handling.  ***9. ”Use Effective Quality Assurance Techniques” -*** Utilizing correct exception handling is a key component of effective quality assurance techniques and if utilized correctly can ensure the software and systems are secure. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.26 | V5002 | PVS-Studio's checker detects issues related to exception handling, ensuring proper error handling and code reliability. |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | CERT C++: ERR56-CPP | Checks for exceptions violating class invariant. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Null Pointer Check** | STD-008-CPP | This standard ensures that pointers are checked for null or nullptr before dereferencing to prevent null pointer dereference errors. |

| **Noncompliant Code** |
| --- |
| In this example, there's no null pointer check, which can lead to a null pointer dereference. |
| int\* ptr = nullptr;  // no check to validate pointer  int value = \*ptr; |

| **Compliant Code** |
| --- |
| To comply with the Null Pointer Checks standard, always check pointers for null or nullptr before dereferencing. |
| int\* ptr = nullptr;  if (ptr != nullptr) {  int value = \*ptr; // pointer is not null pointer  } |

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

| **Principles(s):**  ***8. "Practice Defense in Depth" -*** principle aligns with this standard by emphasizing the need to check pointers for null before dereferencing, adding an extra layer of security. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CPPCheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | nullPointerDereference | CppCheck's nullPointerDereference checker identifies potential null pointer dereference issues in C++ code. |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | CHECKED\_RETURN  NULL\_RETURNS  REVERSE\_INULL  FORWARD\_NULL | Finds instances where a pointer is checked against NULL and then later dereferenced |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **File Input and Output** | STD-009-CPP | This standard ensures that file input and output operations are performed securely to prevent unauthorized access, data corruption, or unintended file changes. It promotes best practices for reading from and writing to files in a safe and controlled manner. |

| **Noncompliant Code** |
| --- |
| Opening and writing to a file without error checking or proper resource management. |
| #include <iostream>  #include <fstream>  int main() {  std::ofstream file("data.txt");  file << "Hello, world!";  return 0;  } |

| **Compliant Code** |
| --- |
| Securely opening, writing to, and closing a file with proper error handling. |
| #include <iostream>  #include <fstream>  int main() {  std::ofstream file("data.txt");  // validate file is open, handle error if not  if (!file.is\_open()) {  std::cerr << "Failed to open file." << std::endl;  return 1;  }  file << "Hello, world!";  // validate write to file, error if not  if (file.fail()) {  std::cerr << "Failed to write to file." << std::endl;  file.close();  return 1;  }  // close file when done with it  file.close();  return 0;  } |

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

| **Principles(s):**  ***1. “Validate Input Data” -*** Relates to this principle as all data being passed into the system should be validated and verified before being utilized.  ***7. “Sanitize Data Sent to Other Systems” -*** All data should be sanitized before writing to a file to ensure incorrect data types, values, etc. are not utilized.  ***8. "Practice Defense in Depth" -***principle aligns with this standard by emphasizing secure file I/O practices as part of a layered security approach. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call. |
| [Parasoft](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-FIO51-a | Ensures resources are freed. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Code Commenting** | STD-010-CPP | The Code Comments standard ensures that code is well-documented with clear and informative comments to improve code readability and maintainability. |

| **Noncompliant Code** |
| --- |
| In this code, there are no comments to explain the purpose or functionality of the code. |
| int x = 42; |

| **Compliant Code** |
| --- |
| To comply with the Code Comments standard, provide clear and informative comments to explain the code's purpose and functionality. |
| int x = 32; // Initialize x to default value of 32, freeze point in F |

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

| **Principles(s):**  ***3. "Architect and Design for Security Policies" -*** principle relates to this standard by emphasizing the importance of clear and informative comments to improve code readability and maintainability, which is essential for secure code maintenance. |
| --- |

**Threat Level**

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

**Automation**

Currently, there are no automation tools available to confirm if comments are included in the completed code base. Code Reviews should be completed to ensure proper and informative comments are being created to support the system.

### 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 plays a crucial role in enforcing and ensuring compliance with the standards outlined in this policy within Green Pace's well-established DevOps process and infrastructure. To integrate these security standards effectively, we will align automation with the DevSecOps diagram, leveraging each phase of the software development lifecycle (SDLC) to enhance security practices.

All plans developed in the pre-production phases, Assess and Plan through Design, should utilize this policy to ensure that coding principles and coding standards are being considered and planned for in accordance with Green Pace policies. Once development begins, testing should be completed utilizing the proposed tools in the policy to validate security compliance with this policy. Once validation is completed, the project can move into production, executing the back-end testing requirements needed to validate it passes penetration testing so it can be configured and deployed safely. As the system is deployed, continuous testing and monitoring will be conducted to block attempted attacks and react accordingly to mitigate the attacks and prevent future attacks.

By executing this automation plan, we are able to successfully transition the current Green Pace DevOps pipeline into a DevSecOps pipeline.

### 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 | Possible | Medium | High | 4 |
| STD-002-CPP | High | Unlikely | Low | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | High | 5 |
| STD-004-CPP | High | Likely | High | High | 5 |
| STD-005-CPP | High | Possible | Medium | High | 4 |
| STD-006-CPP | Low | Unlikely | Low | Low | 1 |
| STD-007-CPP | High | Likely | Medium | High | 4 |
| STD-008-CPP | Medium | Possible | Low | Medium | 3 |
| STD-009-CPP | High | Possible | Medium | Medium | 3 |
| STD-010-CPP | Low | Unlikely | Low | Medium | 2 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | **What it is:** Encryption at rest is the process of securing data when it is stored on physical or digital storage devices, such as hard drives, databases, or cloud storage.  **How and why the policy applies:** All sensitive data, including customer information, employee records, and proprietary information, must be encrypted when stored on any medium. This policy applies to all data repositories within Green Pace, including servers, databases, and cloud storage solutions. Encryption at rest helps safeguard data from unauthorized access in case of physical theft, unauthorized access, or data breaches. |
| Encryption at flight | **What it is:** Encryption in flight, also known as data in transit encryption, is the process of securing data when it is transmitted over a network or during communication between systems.  **How and why the policy applies:** All communication channels, both internal and external, that involve sensitive data must employ encryption mechanisms to protect data during transmission. This policy applies to email communications, web transactions, and any other data transfer methods. Encryption in flight ensures that data remains confidential and integral while being transmitted, mitigating the risk of eavesdropping or interception by malicious actors. |
| Encryption in use | **What it is:** Encryption in use refers to the encryption of data while it is actively being processed or used by applications or services.  **How and why the policy applies:** When sensitive data is being processed or used within applications or services, it must be encrypted to prevent unauthorized access or leakage. This policy applies to software applications, cloud services, and any other systems that handle sensitive data in real-time. Encryption in use ensures that data remains protected throughout its entire lifecycle, from storage and transmission to processing. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | **What it is:** Authentication is the process of verifying the identity of users or systems trying to access Green Pace's resources.  **How and why the policy applies:** All users and systems accessing Green Pace's resources must undergo a robust authentication process to ensure that only authorized entities gain access. This policy applies to user logins, including two-factor authentication when necessary, and the use of strong, unique passwords. Authentication prevents unauthorized access to sensitive data and systems. |
| Authorization | **What it is:** Authorization is the process of granting or denying access to specific resources or functionalities based on a user's identity and permissions.  **How and why the policy applies:** Green Pace must implement fine-grained authorization controls to restrict user access to only the resources and actions necessary for their roles. This policy applies to controlling the user's level of access to data, applications, and systems. Authorization ensures that users can only perform actions and access data relevant to their job responsibilities, reducing the risk of data breaches and unauthorized activities. |
| Accounting | **What it is:** Accounting, also known as auditing, involves tracking and monitoring user activities, changes to the database, additions of new users, and file access by users.  **How and why the policy applies:** Green Pace must maintain detailed logs and records of all user activities and system events, including logins, data modifications, and file access. This policy applies to recording and storing logs securely for auditing purposes. Accounting helps in the detection of suspicious or unauthorized activities, aids in incident response, and ensures compliance with regulatory requirements. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 11/10/2023 | Coding Principles and Standards | Matthew Bandyk |  |
| 1.2 | 12/2/2023 | Policy Finalized | Matthew Bandyk |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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