**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 | Input validation to ensure that it is properly formatted. This helps prevents SQL injection. |
| 1. Heed Compiler Warnings | They are potential security issues or bugs. Addressing these warning can help prevent vulnerabilities that can be exploited. |
| 1. Architect and Design for Security Policies | Defining clear security policies and enforcing them. |
| 1. Keep It Simple | Simple clean designs makes it easier to analyze security flaws. |
| 1. Default Deny | Denying all access by default and granting permission when necessary. Helps prevent unauthorized access. |
| 1. Adhere to the Principle of Least Privilege | Giving least amount of privileges to a user or program to avoid and compromising features. |
| 1. Sanitize Data Sent to Other Systems | Going through data to make sure that it is free of any malicious content. |
| 1. Practice Defense in Depth | Layers of defense depending on your needs. |
| 1. Use Effective Quality Assurance Techniques | Reviewing code, testing security, and assessing vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Setting guidelines and best practices to prevent security 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] | Choosing the appropriate date for a variable. It ensures the range of values can be represented is correct for the intended use. It ensures the safety of data by preventing overflow, mathematical errors, and data corruption. |

| **Noncompliant Code** |
| --- |
| Using ‘int” instead of “char” for storing the character “A” |
| int letter = “A”; |

| **Compliant Code** |
| --- |
| Using ‘int’ for loop counter since it is expected for range of values. The choice aligns with using data types that match expected range of data, |
| int main() {  for (int I = 0; I < 10; i++) {  std::cout << “Loop iteration: “ << I << std::endl;  }  return 0;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Static Code Analysis | 1.0.0 | Data Type Usage | Scans the source code and highlights potential mismatches and gives of potential corrections. |
| Compiler with Strict Type Checking | [Insert text.] | Type Safety | Flags any instance of data that is mismatched or not being used properly. |
| IDE Code Analysis Plugin | 2.3.0 | Data Type Consistency | It integrates into the development environment and gives feedback in real time on data type consistency. It can highlight the code where it may not follow the intended use of the code. |
| Listing Tool for Data Type Conventions | 1.1.0 | Data Type Conventions | This tool enforces data type conventions. Checks for consistency throughout based off standards and coding guidelines. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | This standard is in place to make sure that all data values are used and managed appropriately. This standard applies the proper use of data to prevent SQL injections. It helps maintain the codebase through consistent data value. |

| **Noncompliant Code** |
| --- |
| The following snippet does not validate the input received. It does not check the size, or any other potential validation can cause issues like buffer overflow. |
| int main() {  std::string password;  std::cout << "Enter your password: ";  //No validation of input  std::cin >> password;  std::cout << "Password entered: " << password << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| The input validation is conducted by checking the length of the password, and other characters in the input. Since we do this we can prevent buffer overflow if the password input is too long. |
| int main() {  std::string password;  std::cout << "Enter your password: ";    // Innput is properly validated and within bounds  std::getline(std::cin, password);  while (password.length() > 50) {  std::cout << "Password is too long. Please enter a password with at most 50 characters: ";  std::getline(std::cin, password);  }  std::cout << "Password entered: " << password << std::endl;  return 0;  } |

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

| **Principles(s):** Validate and Input Date  We enforce data validation of input values to make sure that the code only receives acceptable inputs. By only allowing acceptable inputs it prevents security vulnerabilities such as buffer overflow or injection attacks if they go unchecked. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Static Analyzer | 2.0.1 | Input Validation | This tool checks for proper validation of data that is input. It makes sure that the program is using the input correctly and securely. |
| Dynamic Scanner | 1.5.0 | Buffer Overflow | IT detects the program as it executes and identifies any buffer overflow that may be occurring. |
| Code Linter | 3.2.0 | Coding standard | It checks the code that it is written up to coding standards. |
| Security Scanner | 4.1.2 | Vulnerability Assessment | It can detect potential vulnerabilities against the codebase. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | This standard ensures the correct usage and manipulation of string objects. Doing so will prevent vulnerabilities such as memory leaks and data corruption. |

| **Noncompliant Code** |
| --- |
| The code fails to perform the boundary checking when reading the input. This can lead to buffer overflow. |
| int main() {  char buffer[10];  // Read input without boundary checking  std::cin >> buffer;  // Potential buffer overflow if input exceeds the buffer size  return 0;  } |

| **Compliant Code** |
| --- |
| Using the ‘std::getline()’ function we are able to safely read the input into a string avoiding the buffer overflow. |
| int main() {  std::string input;  // Read input safely into a string  std::getline(std::cin, input);  // No risk of buffer overflow as string handles memory dynamically  return 0;  } |

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

| **Principles(s):** Valid input Data  It assists in the validating of all inputs. This enhances safety and security. It reduces the likeliness of exploits and attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Static Code analysis | 2.0.1 | Buffer Flow Detection | Can check for potential vulnerabilities in the codebase. |
| Dynamic Analysis Tool | 1.5.0 | Memory Leak Detection | Scans the code when it is running for memory leaks and makes sure that memory is being properly managed with the string objects. |
| IDE Plugin | 3.2.0 | Input Validation Checker | This ide integrates into the development and |
| Linting Tool | 1.1.0 | Coding Standards Enforcement | Enforces coding standards related to string usage. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | IT aims to prevent SQL injections vulnerabilities. It does so by adhering to the use of proper input validation. |

| **Noncompliant Code** |
| --- |
| SQL query is created by combining user input into the query string. This way is vulnerable to SQL injections because attackers can enter SQL commands into the input. |
| int main() {  std::string username;  std::cout << "Enter your username: ";  std::cin >> username;  //Vulnerable to SQL injection  std::string query = "SELECT \* FROM users WHERE username='" + username + "'";  std::cout << "Executing query: " << query << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| Using parameterized queries, our input values are sent separately to the SQL query string, preventing the SQL injection attacks. It is a compliant standard to prevent SQL injections vulnerabilities. |
| int main() {  std::string username;  std::cout << "Enter your username: ";  std::cin >> username;  //Using parameterized query  std::stringstream query;  query << "SELECT \* FROM users WHERE username=?";  std::cout << "Executing query: " << query.str() << std::endl;  return 0;  } |

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

| **Principles(s):** Valid Input Date  This standard and principle align because they are both make aware the importance of validating input. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Static Code Analysis Tool | 2.0.1 | SQL Injection Scanner | It identifies SQL injection vulnerabilities. |
| Dynamic Security Scanner | 1.5.0 | Database vulnerability Scanner | Scans web apps and DB systems for SQL injection vulnerabilities when it is running. |
| Security Testing Framework | 3.2.0 | Input Validation Testing | Automates testing of the input validation mechanisms such as parameterized queries. |
| Code Review Tool | 1.1.0 | SQL Injection Reviewer | Assist in reviewing the code for potential SQL vulnerabilities through the highlighting of uncertain coding practices. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | This standard focuses on memory leaks, buffer overflow and dangling pointers. It creates practices like bounds checking, proper memory allocation and deallocation. It also ensures memory safety and prevents exploitation through the safe use of data structures. |

| **Noncompliant Code** |
| --- |
| The code here fails to deallocate dynamically allocated memory after it is not needed. This results in memory leaks. Thus it will cause inefficient memory usage. |
| int main() {  int\* ptr = new int; // Allocate memory for an integer  // Noncompliant code: No deallocation of dynamically allocated memory  // Memory leak occurs as the allocated memory is not freed  return 0;  } |

| **Compliant Code** |
| --- |
| By correctly using the delete operator we can deallocate the dynamically allocated memory when it is no longer needed. Make the memory management more efficient. |
| int main() {  int\* ptr = new int; // Allocate memory for an integer  // Compliant code: Deallocation of dynamically allocated memory  delete ptr; // Free the allocated memory  return 0;  } |

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

| **Principles(s):** Keep it Simple  To me keeping it simple with a clean design can prevent memory related vulnerabilities. If the complexity of the codebase is minimal but effective, can lower the potential memory related vulnerabilities. The more complex the more likely these can occur. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.15.0 | Memory Leak Checker | It is a memory debugger and leak detection and profiling. |
| Cppcheck | 2.3 | Buffer Overflow Detection | It is a static analysis tool that helps detect buffer overflow vulnerabilities. |
| SonarQube | 8.9.0 | Custom Rules for Code Quality | It provides static code analysis to detect bugs and code smells that ensure code quality. |
| Coverity | 2021.03 | Secure Coding Standard Checks | It checks for adherence and potentials security vulnerabilities ensuring best practices. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | It assumes the state of the program during the execution of it. IT helps developers validate assumptions about how the program behaves, input parameters and the internal state of it. Checkpoints are created to catch unexpected errors during the runtime. |

| **Noncompliant Code** |
| --- |
| In this code we don’t not create any assertions to check if the divisor is 0 before executing the division operator. Creating a potential runtime error if it is. |
| int divide(int dividend, int divisor) {  // Noncompliant code: Lack of assertion to check divisor not being zero  return dividend / divisor;  }  int main() {  int result = divide(10, 0);  // Code continues...  return 0;  } |

| **Compliant Code** |
| --- |
| Here we add a step where the assertion added checks to make sure that the divisor does not = 0 first. If it doesn’t letting the code continue. |
| int divide(int dividend, int divisor) {  // Compliant code: Assertion to check divisor not  assert(divisor != 0); being zero  return dividend / divisor;  }  int main() {  int result = divide(10, 2);  // Code continues  return 0;  } |

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

| **Principles(s):** Architect and Design for Security Policies  The coding standard of assertions relates to this principle as it is an important of how the architect of the code will be created. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| GTest | 1.11.0 | Assertion Macros | Google test has a set of assertion macros that are used for writing tests. The macros allow developers to create assertions to the expectation result of their code |
| Boost.Test | 1.77.0 | BOOST\_CHECK macros | This tool provides Boost libraries to perform checks and validation during unit testing. |
| CppUTest | 4.0 | Assertion Macros | This tool provides developres assertion macros and utilities that ensure the correctness of their embedded systems software components. |
| Catch2 | 3.0.0 | BDD-style assertions | Behavior drive development assertions is a header only C++ testing framework. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exceptions are standards that emphasize the proper handling usage of exceptions in the software development stage. They are mechanisms used to handle conditions or errors that may occur during program execution. The standard ensures that exceptions are utilized effectively and consistently throughout the codebase. |

| **Noncompliant Code** |
| --- |
| Here the code has a pretty generic catch block that without a specific exception handling, can result in obscuring the details. |
| try {  int result = divide(10, 0); // Division by zero  }  catch (...) {  // Noncompliant code: Generic catch block without specific exception handling  std::cerr << "An exception occurred" << std::endl;  } |

| **Compliant Code** |
| --- |
| In this code exceptions are used to handle exception conditions. One being the division of zero. In the main function try-catch block is used to catch specific exceptions. In this code being the invalid argument. The exceptions here are properly handled with error messages. |
| int divide(int dividend, int divisor) {  if (divisor == 0) {  throw std::invalid\_argument("Divisor cannot be zero");  }  return dividend / divisor;  }  int main() {  try {  int result = divide(10, 0); // Division by zero  std::cout << "Result: " << result << std::endl;  }  catch (const std::invalid\_argument& ex) {  // Compliant code: Specific exception handling for invalid argument  std::cerr << "Exception caught: " << ex.what() << std::endl;  }  return 0;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques  Proper exception handling ensures reliability and security of software systems. Effective quality assurance techniques aim to deliver robust and secure software products, and by exceptions standards they aim as well to maintain proper handling and usage of exceptions. Exceptions create a way where errors occur developers can catch these during program execution and continue to correct and make the software better. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2021.03 | Exception Handling Checks | The tool provides static analysis, to detect issues related to exception handling. |
| SonarQube | 8.9.0 | Exception Handling Rules | Offers static code analysis that identifies issues with any exception handling. |
| CppCheck | 2.3 | Exception checker | Static analysis tool checking proper exception handling detecting potential issues in the code. |
| ReSharper c++ | 2021.3 | Exception Usage Analysis | Another static analysis tool that checks for proper exception handling. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Cryptographic Usage | [STD-008-CPP] | This standard focuses on making sure of the proper usage of cryptographic algorithms and protocols. Its aim is to promote secure practices. Its goal of this standard is to make sure that the we are following bests practices while helping to protect sensitive information and maintain security. |

| **Noncompliant Code** |
| --- |
| This code snippet uses the MD5 hash function to hash a password. The MD5 though is considered s=insecure because of its inability to prevent collision attacks and pre image attacks. |
| void insecureHash(const char\* input, char\* output) {  //Using insecure MD5 hash function  MD5((const unsigned char\*)input, strlen(input), (unsigned char\*)output);  }  int main() {  char hash[MD5\_DIGEST\_LENGTH];  insecureHash("password123", hash);  return 0;  } |

| **Compliant Code** |
| --- |
| In this snippet we use the SHA-256 hash function from the OpenSSL library. Using this hash function we practice secure Cryptographic coding standards. The function takes the input string and stores it into a hash array. |
| void secureHash(const char\* input, unsigned char\* output) {  // Compliant code: Using SHA-256 hash function  SHA256\_CTX context;  SHA256\_Init(&context);  SHA256\_Update(&context, input, strlen(input));  SHA256\_Final(output, &context);  }  int main() {  const char\* password = "password123";  unsigned char hash[SHA256\_DIGEST\_LENGTH];  secureHash(password, hash);  // Output the hash (hexadecimal representation)  for (int i = 0; i < SHA256\_DIGEST\_LENGTH; ++i) {  printf("%02x", hash[i]);  }  printf("\n");  return 0;  } |

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

| **Principles(s):** Default Deny  Cryptographic usage and Default Deny align as they both aim to control who can access what. In doing so it protects data that isn’t for everybody. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| OpenSSL | 1.1.1 | Crypto Checker | This tool provides certain functions that can encrypt and hash. |
| Botan | 2.18.0 | Cryptography API | Botan is a library that can support modern cryptographic algorithms and protocols. |
| Crypto++ | 8.6.0 | Crypto Library | It is a library that offers key derivation and digital signatures. |
| Libsodium | 1.0.18 | Cryptographic tool | It is a library that offers encryption and hashing. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Logging and Monitoring | [STD-009-CPP] | Effectively logging practices and standards can capture real time events. Specifically on errors. It is a way where we can respond to certain incidents quickly and effectively. |

| **Noncompliant Code** |
| --- |
| Within this snippet we request a process but fail to add any logging mechanisms. This lead to no records of what actually occurred and what was executed here, |
| // Function to process a request  void processRequest(int requestId) {  //No logging of request processing  std::cout << "Processing request with ID: " << requestId << std::endl;  }  int main() {  int requestId = 123;  processRequest(requestId);  return 0;  } |

| **Compliant Code** |
| --- |
| This code uses a function ‘logToFile’ to append messages, with timestamps where they are logged in a file named ‘application.log’. It checks whether the file can be opened and if not, it will display an error message. |
| void logToFile(const std::string& message) {  std::ofstream logfile("application.log", std::ios::app);  if (!logfile.is\_open()) {  std::cerr << "Error: Unable to open log file!" << std::endl;  return;  }  logfile << "[" << std::asctime(std::localtime(&(std::time(nullptr)))) << "] " << message << std::endl;  }  void processRequest(int requestId) {  std::cout << "Processing request with ID: " << requestId << std::endl;  logToFile("Request processed with ID: " + std::to\_string(requestId));  }  int main() {  processRequest(123);  return 0;  } |

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

| **Principles(s):** Practice Defense in Depth  Defense in Depth makes sure that we are creating a way that the code is being tracked at all stages of its deployment. By placing logging techniques at each stage of its life cycle, we can monitor how it behaves throughout the process. Ultimately we look to prevent and block any vulnerabilities no matter what stage it is at in development. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Splunk | 8.3 | Log Monitoring and Analysis | This tool provides real time monitoring, analysis and visualization of the log data. |
| ELK Stack | 7.15 | Log Management | This tool allows to gather all amount of log data from your infrastructure and combines it into a unified log which from there you can analyze and search through it in real time. |
| Log4cpp | 1.1.3 | Logging Library | C++ library used for logging configuration. |
| Boost.log | 1.76.0 | Loggin Framework | It is another logging library that offers extensive features to developers like filtering mechanisms. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Handling | [STD-010-CPP] | We use this standard so we can focus on creating guidelines and best practices for errors that occur. By being able to handle errors effectively, we can identify these errors quicker, and minimize them appropriately. |

| **Noncompliant Code** |
| --- |
| In this code we attempt to open a file but apply no error handling standards. If the file fails to open there is no handling of what should happen if this occurs. If this error isn’t reported back we have no ability or are even aware that we need to handle an error that occurred. |
| void writeToFile(const std::string& filename, const std::string& content) {  std::ofstream file(filename);  if (!file.is\_open()) {  //No error handling for file opening failure  std::cerr << "Error: Unable to open file " << filename << std::endl;  return;  }  file << content;  file.close();  }  int main() {  std::string filename = "example.txt";  std::string content = "Hello, world!";  writeToFile(filename, content);  return 0;  } |

| **Compliant Code** |
| --- |
| In this updated code when we try to write to the file after it is opened, we add an additional line of code where the programs return with an error rather than attempting to execute the writeToFile function. This makes sure that if an error does occur it is returned to us and does not continue to attempt and complete the function. It makes us aware of an issue that we can fix before moving on. |
| void writeToFile(const std::string& filename, const std::string& content) {  std::ofstream file(filename);  if (!file.is\_open()) {  std::cerr << "Error: Unable to open file " << filename << std::endl;  //Handle error by returning early  return;  }  file << content;  file.close();  }  int main() {  std::string filename = "example.txt";  std::string content = "Hello, world!";  writeToFile(filename, content);  return 0;  } |

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

| **Principles(s):** Use effective quality assurance techniques.  We implement proper error handling techniques throughout the code, so it can align with making sure that we are producing quality code. Developers can make sure that errors are caught throughout the process instead after the fact. Quality Assurance applies similar techniques where tests are run throughout simulating errors that may occur and how they are going to be handled if they do coccur. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.15.0 | Memcheck | This tool detects memory management often issues in error handling. Such examples include memory leaks and buffer overflows. |
| Coverity | 2021.03 | Error Handling | The tool identifies issues related to error handling practices such as inconsistent error handling approaches. |
| Clang Static Analyzer | 13.0.0 | Clang-analyzer | This tool detects unchecked return values to prevent unhandled errors. |
| PVS-Studi | 7.13 | V649 | It checks for issues related to error handling. Examples include not attending to error codes returned by functions |

### 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.

Using the DevSecOps diagram already in place we will look at enhancing the Developers tools through automation, by using tools that automatically check their code for security as they write it. A certain tool that we can implement to do so is SonarQube. This tool integrates with your development environment and scans the code for vulnerabilities. This way we can reassess the beginning of the pre-production. We can aim to have no backlog of any potential threats and address them now.

For the Security portion we can set rules that automatically stope the code from moving forward if it doesn’t meet security standard such automated tools that can assist with this type of response can include JetBrains IntelliJ IDEA, in which their security plugins give feedback immediately to developers on potential security issues as they write their code.

On the operations side we can use scripts and tools to set up and manage the apps environments making sure that it can always be configured securely. One way we can attempt to implement this into the operations workflow through automation is by adopting or integrating the use of Prism Cloud that can regularly check and enforce compliance and security policies.

### 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 | Low | High | 3 |
| STD-003-CPP | High | Likely | Low | High | 3 |
| STD-004-CPP | High | Very Likely | High | High | 4 |
| STD-005-CPP | High | Likely | High | High | 3 |
| STD-006-CPP | High | Likely | High | High | 3 |
| STD-007-CPP | High | Likely | Medium | High | 3 |
| STD-008-CPP | High | Likely | High | High | 3 |
| STD-009-CPP | Medium | Likely | High | High | 3 |
| STD-010-CPP | High | Very likely | Medium | High | 3 |

### 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 | This protects any data that is not active that is stored on the storage device or network. Any data that is handled needs to be encrypted whether it is used or not. This helps protect any sensitive data that may be out there in your database. |
| Encryption at flight | This is encrypting and securing data that is moving from point a to point b though the network. It is safe to encrypt data being sent to other networks especially if these networks are not know to you. Any data sent over the network is at risk of being intercepted. |
| Encryption in use | We encrypt data that we are currently using. This applies especially if you’re handling sensitive data reducing the risk of exposure to someone that should not have access to it. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is making sure that we verify that the identity of the user attempting to access something. We do this through several features such as passwords, and tokens. This makes it so only the people allowed to see a certain piece of data has access to it only. |
| Authorization | It is the process of giving access to someone over a secure system. Since we are first authenticated, and then depending on the type of authentication we are given we can be granted specific authorization in the organization. |
| Accounting | This is referencing the tracking of all user activity. Each user’s keystroke is tracked and logged. All maintenance of user creation and manipulation is recorded as well. This essentially creates a trail and can be audit in case of any incident that can occur. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 02/01/204 | First Draft | Joseph Valle | Joseph Valle |
| 2.0 | 2/18/2024 | Final | Joseph Valle | Joseph Valle |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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