**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 is probably the biggest filter for vulnerabilities and exploitations. As a programmer, one must always be suspicious and inquisitive of data that comes from untrusted/unknown sources. This data must be checked for malicious content. If one fails to do this, protecting the system against attacks will be exponentially more difficult. |
| 1. Heed Compiler Warnings | A programmer should always compile code using the highest warning level available. It can be tempting to ignore the warnings since the program will still run, but ignored warnings can compound over time and introduce unexpected behavior, and even crashes, during runtime. |
| 1. Architect and Design for Security Policies | No architecture is a one-size-fits-all solution. It is therefore imperative to design and customize your system architecture so that it complements the way you and the users will be using the system. |
| 1. Keep It Simple | This principle is pretty self-explanatory. Superfluous complexity will just make maintaining and updating the system needlessly difficult. Make sure things are organized and commented diligently. |
| 1. Default Deny | This principle maintains that all users and entities will be, by default, denied from accessing a given resource, and then the system will then do the work needed to be done to determine if the status should remain “denied” or if the entity in question actually has permission to access the resource in question. |
| 1. Adhere to the Principle of Least Privilege | Processes should not be given more privilege than they need. This is essentially a way of mitigating the damage in the case that a process has been compromised. |
| 1. Sanitize Data Sent to Other Systems | This principle is essentially the reverse of the “Validate Input Data” principle. This time, we want to verify that the information *our system* is *giving* to another system doesn’t contain any information that could be used to harm our system. |
| 1. Practice Defense in Depth | This principle is basically to always have a backup plan. If one security measure fails to neutralize a threat, there ideally should be several laying in wait that can pick up the slack. The goal is security redundancy. |
| 1. Use Effective Quality Assurance Techniques | This principle maintains that in order to have the upmost confidence in a newly created system, it must be heavily peer reviewed and tested in a variety of ways. Doing this can eliminate exploits before they even happen. Prevention being better than cure is the philosophy of this principle. |
| 1. Adopt a Secure Coding Standard | Following a secure standard will give direction and guidance in the task of creating secure code. Rather than trying to go about the process unstructured, doing this is the most efficient way to ensure that all the t’s are crossed and all I’s are dotted. |

### 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** | **Type Consistency** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Use consistent types to avoid losing information. |

| **Noncompliant Code** |
| --- |
| Mixing incompatible data types like int and float can lead to unexpected errors and loss of info/precision |
| int x = 17;  float y = 5.75;  int result = x + y; |

| **Compliant Code** |
| --- |
| Adding a float variable ensures proper type compatibility |
| int x = 17;  float y = 5.75;  float result = x + y; |

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

| **Principles(s):** DCL31-C. Declare identifiers before using them  This principle maps to the standard because declaring the type of an identifier is not only required, but it helps keep variables organized and reduces likely hood of mismatching types when wanting to use variables with one another. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.03 | Type-specifier function-return-type  implicit-function-declaration  undeclared-parameter | Fully Checked |
| Axivion Bauhaus suite | 2.2.0 | CertC-DCL31 | Fully Implemented |
| Clang | 3.9 | -Wimplicit-int |  |
| Compass/ROSE |  |  |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Nonzero divisor** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Dividing by zero is mathematically meaningless and will cause errors. |

| **Noncompliant Code** |
| --- |
| Dividing by Zero is not allowed in C++ |
| int a = 17;  int b = 0;  int result = e / f; |

| **Compliant Code** |
| --- |
| Avoiding dividing by Zero |
| int a = 17;  int b = 2;  if (b != 0) {  int result = e / f;  cout << result << endl;  } else {  cout << "Cannot divide by zero" << endl;  } |

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

| **Principles(s):** INT33-C. Ensure that division and remainder operations do not result in divide-by-zero errors  This principle maps to the standard directly and discusses divide-by-zero errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | int-division-by-zero  int-modulo-by-zero | Fully Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT33 |  |
| CodeSonar | 7.4p0 | LANG.ARITH.DIVZERO  LANG.ARITH.FDIVZERO | Division by zero Float Division By Zero |
| Compass/ROSE |  |  | Can detect some violations of this rule (In particular, it ensures that all operations involving division or modulo are preceded by a check ensuring that the second operand is nonzero.) |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Input Validation** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Data coming from unknown sources can be dangerous, it is important to make sure the content of the data is suitable with your desires. |

| **Noncompliant Code** |
| --- |
| Improper input validation because all this code does is check if a username is greater than 10 chars, despite the fact that the system requirements for a proper username are more complex than just length. |
| #include <iostream>  #include <string>  bool isValidUsername(std::string username) {  // Improper input validation  if (username.length() > 10) {  return false;  }  return true;  }  int main() {  std::string username;  std::cout << "Enter username: ";  std::cin >> username;  if (isValidUsername(username)) {  std::cout << "Username is valid." << std::endl;  } else {  std::cout << "Invalid username." << std::endl;  }  return 0;  } |

| **Compliant Code** |
| --- |
| Uses regular expressions to verify that the received data fits the system requirements and standards. |
| #include <iostream>  #include <string>  #include <regex>  bool isValidUsername(std::string username) {  // Proper input validation using regular expressions  std::regex regex("^[a-zA-Z0-9\_-]{3,16}$");  return std::regex\_match(username, regex);  }  int main() {  std::string username;  std::cout << "Enter username: ";  std::cin >> username;  if (isValidUsername(username)) {  std::cout << "Username is valid." << std::endl;  } else {  std::cout << "Invalid username." << std::endl;  }  return 0;  } |

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

| **Principles(s):** FIO30-C. Exclude user input from format strings  This principle maps to the standard because adhering to it is part of input validation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 |  | Supported via stubbing/taint analysis |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO30 | Partially implemented |
| CodeSonar | 7.4p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| Compass/ROSE |  |  |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **SQL Injection Protection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Neglecting to protect from potential SQL Injection attacks can lead to someone gaining unauthorized access to your system. |

| **Noncompliant Code** |
| --- |
| Vulnerable to SQL injection since code directly concatenates the username input into the SQL query without proper sanitization of parameterization. |
| #include <iostream>  #include <mysql/mysql.h> // Assuming MySQL database is used  int main() {  MYSQL\* connection;  mysql\_init(connection);  if (!mysql\_real\_connect(connection, "localhost", "user", "password", "database", 0, NULL, 0)) {  std::cerr << "Failed to connect to database: " << mysql\_error(connection) << std::endl;  return 1;  }  std::string username;  std::cout << "Enter username: ";  std::cin >> username;  std::string query = "SELECT \* FROM users WHERE username = '" + username + "'";  if (mysql\_query(connection, query.c\_str())) {  std::cerr << "Failed to execute query: " << mysql\_error(connection) << std::endl;  return 1;  }  MYSQL\_RES\* result = mysql\_store\_result(connection);  // Process the result  // ...  mysql\_free\_result(result);  mysql\_close(connection);  return 0;  } |

| **Compliant Code** |
| --- |
| Properly protects against SQL injections by checking input |
| #include <iostream>  #include <mysql/mysql.h> // Assuming MySQL database is used  int main() {  MYSQL\* connection;  mysql\_init(connection);  if (!mysql\_real\_connect(connection, "localhost", "user", "password", "database", 0, NULL, 0)) {  std::cerr << "Failed to connect to database: " << mysql\_error(connection) << std::endl;  return 1;  }  std::string username;  std::cout << "Enter username: ";  std::cin >> username;  std::string query = "SELECT \* FROM users WHERE username = ?";  MYSQL\_STMT\* stmt = mysql\_stmt\_init(connection);  if (!mysql\_stmt\_prepare(stmt, query.c\_str(), query.length())) {  MYSQL\_BIND param;  memset(&param, 0, sizeof(param));  param.buffer\_type = MYSQL\_TYPE\_STRING;  param.buffer = (void\*)username.c\_str();  param.buffer\_length = username.length();  mysql\_stmt\_bind\_param(stmt, &param);  mysql\_stmt\_execute(stmt);  // Process the result  // ...  }  mysql\_stmt\_close(stmt);  mysql\_close(connection);  return 0;  } |

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

| **Principles(s):** IDS00-J. Prevent SQL injection  These principle maps to the standard directly. It discusses the prevention of SQL injection |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting checker | Trust and security errors (see Chapter 8) |
| CodeSonar | 7.4p0 | JAVA.IO.INJ.SQL | SQL Injection (java) |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Memory Management** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | If the memory becomes corrupted during the runtime of your program, the program will crash. |

| **Noncompliant Code** |
| --- |
| Code that uses excessive memory allocation |
| #include <iostream>  int\* createArray(int size) {  int\* arr = new int[size];  return arr;  }  int main() {  int size = 1000000000; // Huge size intentionally to demonstrate improper memory handling  int\* arr = createArray(size);  // Use the array...  delete[] arr; // Proper deallocation  return 0;  } |

| **Compliant Code** |
| --- |
| By choosing to use a vector rather than a bare array, this code is utilizing standard library memory management. |
| #include <iostream>  #include <vector>  std::vector<int> createVector(int size) {  std::vector<int> vec(size);  return vec;  }  int main() {  int size = 1000000000; // Large size  std::vector<int> vec = createVector(size);  // Use the vector...  return 0;  } |

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

| **Principles(s):** MEM34-C. Only free memory allocated dynamically  This principle maps to the standard because adhering to it will prevent memory corruption. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | Invalid-free | Fully Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM34 | Can detect memory deallocations for stack objects |
| Clang | 3.9 | Clang-analyzer-unix.Malloc | Checked by clang-tidy; can detect some instances of this rule, but does not detect all |
| CodeSonar | 7.4p0 | ALLOC.TM | Type Mismatch |

#### Coding Standard 6

| **Coding Standard** | **Label** |  |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertion Coding Standard |

| **Noncompliant Code** |
| --- |
| If the assert fails, program execution will halt without much explanation or a useful error message. |
| #include <cassert>  int divide(int numerator, int denominator) {  assert(denominator != 0);  return numerator / denominator;  }  int main() {  int num = 10;  int denom = 0; // Improper input  int result = divide(num, denom);  // Rest of the code...  return 0;  } |

| **Compliant Code** |
| --- |
| If the assert fails, the message “Denominator should not be zero” appears. |
| #include <cassert>  int divide(int numerator, int denominator) {  if (denominator == 0) {  assert(false && "Denominator should not be zero");  // Handle the error condition gracefully...  }  return numerator / denominator;  }  int main() {  int num = 10;  int denom = 0; // Improper input  int result = divide(num, denom);  // Rest of the code...  return 0;  } |

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

| **Principles(s):** DCL03-C. Use a static assertion to test the value of a constant expression  This maps to the standard because it is discussing assertion best practices |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.4p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compass/ROSE |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Exception Handling** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Proper exception handling allows your code to be robust and respond gracefully to errors rather than crashing. |

| **Noncompliant Code** |
| --- |
| The exception handling below poses a security risk because the error message potentially reveals sensitive information about the system. |
| #include <iostream>  void processUserInput(int pin) {  if (pin != 1234) {  throw "Invalid PIN!";  }  // Process the user input  }  int main() {  try {  int pin;  std::cout << "Enter PIN: ";  std::cin >> pin;  processUserInput(pin);  std::cout << "Access granted!" << std::endl;  } catch (const char\* error) {  std::cerr << "Error: " << error << std::endl;  }  return 0;  } |

| **Compliant Code** |
| --- |
| The exception handling below is compliant with security standards because the error message is informative without being too revealing. |
| #include <iostream>  #include <stdexcept>  class AccessDeniedException : public std::runtime\_error {  public:  AccessDeniedException() : std::runtime\_error("Access denied!") {}  };  void processUserInput(int pin) {  if (pin != 1234) {  throw AccessDeniedException();  }  // Process the user input  }  int main() {  try {  int pin;  std::cout << "Enter PIN: ";  std::cin >> pin;  processUserInput(pin);  std::cout << "Access granted!" << std::endl;  } catch (const std::exception& ex) {  std::cerr << "Error: " << ex.what() << std::endl;  }  return 0;  } |

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

| **Principles(s):** ERR51-CPP. Handle all exceptions  This principal maps to the standard directly. It discusses exception handling. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all-early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 |  |
| CodeSonar | 7.4p0 | LANG.STRUCT.UCTCH | Unreachable catch |
| Helix QAC | 2023.1 | C++4035, C++4036, C++4037 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Authentication and Authorization Checks** |
| --- | --- | --- |
| Authentication and Authorization | [STD-008-CPP] | Neglecting to authenticate uses of your system properly and securely will allow malicious actors to gain access to the system. |

| **Noncompliant Code** |
| --- |
| Does not authenticate properly and securely |
| #include <iostream>  #include <unordered\_map>  std::unordered\_map<std::string, std::string> userCredentials = {  {"alice", "password123"}, // Alice's plain-text password (noncompliant)  // Other user credentials...  };  bool isAuthenticated(const std::string& username, const std::string& password) {  auto iter = userCredentials.find(username);  if (iter != userCredentials.end()) {  // Validate password directly against the stored plain-text password (noncompliant)  if (password == iter->second) {  return true;  }  }  return false;  }  void performAuthorizedAction(const std::string& username) {  if (isAuthenticated(username, "password123")) {  // Perform authorized action  std::cout << "Action performed successfully" << std::endl;  } else {  // Allow unauthorized access without handling it properly (noncompliant)  std::cout << "Access granted!" << std::endl;  }  } |

| **Compliant Code** |
| --- |
| Performs authentication in a secure manner |
| #include <iostream>  #include <unordered\_map>  std::unordered\_map<std::string, std::string> userCredentials = {  {"alice", "$2y$10$BkE6Em0u0Zy2jq6bfr2WkOsDFuC3O.ZPh6G6izhUw.z/XVeqm0Q1e"}, // Alice's hashed password  // Other user credentials...  };  bool isAuthenticated(const std::string& username, const std::string& password) {  auto iter = userCredentials.find(username);  if (iter != userCredentials.end()) {  // Validate password against the stored hashed password  if (validatePassword(password, iter->second)) {  return true;  }  }  return false;  }  void performAuthorizedAction(const std::string& username) {  if (isAuthenticated(username, "password123")) {  // Perform authorized action  std::cout << "Action performed successfully" << std::endl;  } else {  // Handle unauthorized access  std::cout << "Unauthorized access" << std::endl;  }  }  bool validatePassword(const std::string& password, const std::string& storedHash) {  // Validate the password by comparing it with the stored hash using a secure hashing algorithm  // Return true if the password matches the stored hash, false otherwise  // ...  } |

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

| **Principles(s):** MSC41-C. Never hard code sensitive information  This maps to the standard because refusing to hard code sensitive information is why there is a need for authentication and authorization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | HARDCODED.AUTH  HARDCODED.DNS  HARDCODED.KEY  HARDCODED.SALT  HARDCODED.SEED | Hardcoded Authentication  Hardcoded DNS Name  Hardcoded Crypto Key  Hardcoded Crypto Salt  Hardcoded Seed in PRNG |
| Helix QAC | 2023.2 | C3122 C++3842 |  |
| Klocwork | 2023.2 | HCC  HCC.PWD  HCC.USER  CXX.SV.PWD.PLAIN  CXX.SV.PWD.PLAIN.LENGTH  CXX.SV.PWD.PLAIN.ZERO |  |
| Parasoft C/C++ test | 2023.1 | CERT\_C-MSC41-a | Do not hard code string literals |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Code Organization** |
| --- | --- | --- |
| Organization | [STD-009-CPP] | Organized code is much more maintainable and as a result much more secure. |

| **Noncompliant Code** |
| --- |
| Code divides one number by another but is not very organized. Does not use sensible variable names and does not use abstraction. |
| Int main(){  double x = 5;  double y = 6;  cout << x / y << endl;  } |

| **Compliant Code** |
| --- |
| Code uses meaningful names for variables and abstracts the mathematical logic away into a function. |
| Int divide(double quotient, double divisor){  if (divisor == 0) return 0;  return quotient/ divisor;  }  Int main(){  cout << divide(5, 6) << endl;  } |

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

| **Principles(s):** MSC04-C. Use comments consistently and in a readable fashion  This maps to the standard because commenting is a major way to keep code organized. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | mmline-comment  sline-comment  sline-splicing  smline-comment | Partially Checked |
| GCC | 4.3.5 |  | Can detect violations of this rule when the -Wcomment flag is used |
| ÉCLAIR | 1.2 | CC2.MSC04 | Fully Implemented |
| Helix QAC | 2023.2 | C3108 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Logging and Auditing** |
| --- | --- | --- |
| Logging and Auditing | [STD-010-CPP] | Logging And Auditing your code execution makes for secure code because it gives an extra layer of transparency for the developer in terms of the activities happening within the program. |

| **Noncompliant Code** |
| --- |
| Code that does insufficient logging, logs to the console which will be overwritten if the console window closes, or eventually when enough content is written to the console, the old content is lost. |
| void logEvent(const std::string& message) {  // Log event to console only without persistence  std::cout << message << std::endl;  }  void performAction() {  // Perform action  logEvent("Action performed successfully");  } |

| **Compliant Code** |
| --- |
| Code handles logging correctly, uses a file for more persistent storage. |
| #include <iostream>  #include <fstream>  void logEvent(const std::string& message) {  std::ofstream logFile("application.log", std::ios::app);  if (logFile.is\_open()) {  logFile << message << std::endl;  logFile.close();  }  }  void performAction() {  // Perform action  logEvent("Action performed successfully");  } |

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

| **Principles(s):** **ERR02-J. Prevent exceptions while logging data**   This maps to the standard because in order to do effective logging, you will need to prevent errors from happening and stopping the logging process. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Mediium | Likely | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | JAVA.DEBUG.LOG | Debug Warning (Java) |
| SonarQube | 9.9 | S106 | Standard outputs should not be used directly to log anything |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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.

In the assess and plan step, one must do security risk assessment. Before starting the development, assess the potential security risks of the application. This will help in understanding what security measures need to be in place. Then define clear security requirements. This will set the stage for what needs to be achieved in terms of security throughout the DevOps process.

In the design step, one must do threat modeling. Identify potential threats and design countermeasures. This will help in understanding the potential attack vectors and how to mitigate them. Next, ensure that the design follows secure design principles like the principle of least privilege, defense in depth, etc.

In the build step, one must practice secure coding practices. Train developers on secure coding practices to prevent common vulnerabilities like SQL injections, cross-site scripting, etc. Next, integrate SAST tools into the CI/CD pipeline. These tools will automatically scan the codebase for security vulnerabilities and provide feedback to developers.

In the verify and test phase, one must practice dynamic application security testing (DAST). Once the application is in a runnable state, use DAST tools to test it for vulnerabilities that can only be found when the application is running. Next, you must check for vulnerabilities in third-party libraries and dependencies. Also, periodically, have security experts try to exploit vulnerabilities in the application. This will provide a real-world assessment of the application's security posture.

Now we transition into production. In the transition and health check step, we need configuration management. Ensure that the configurations for production are secure. This includes things like ensuring default passwords are changed, unnecessary ports are closed, etc. Next we have infrastructure as Code (IaC) Scanning. If you're using IaC tools like Terraform or CloudFormation, scan the code for misconfigurations that could lead to security vulnerabilities.

In the monitor and detect phase, we have Security Information and Event Management (SIEM). Use SIEM tools to aggregate and analyze logs for suspicious activities. Also need to use Intrusion Detection Systems (IDS) to monitor the network and system for malicious activities.

In the respond phase, one needs an incident response plan. This means having a clear plan in place for how to respond when a security incident occurs. This includes things like how to contain the incident, how to communicate with stakeholders, etc.

Lastly, in the maintain and stabilize step, one must do regular patching in order to ensure that the system and software are regularly patched to protect against known vulnerabilities. Periodically, have red teams (attackers) try to exploit vulnerabilities while blue teams (defenders) try to defend against them. This will provide a real-world assessment of the organization's security posture and response capabilities.

### 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 | Low | Unlikely | Low | P3 | L3 |
| STD-002-CPP | Low | Likely | Medium | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Probable | Medium | P12 | L1 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Medium | Likely | High | P6 | L2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest ensures that data is encrypted while stored on a disk, aiming to prevent attackers from accessing the unencrypted information. If a hacker discovers a hard drive with encrypted data and lacks the encryption keys, they would need to decrypt it to access the information. |
| Encryption at flight | Encrypting data during its transfer ensures security. In certain applications, like remote replication, the data might be unencrypted when stored on drive arrays but is encrypted when transmitted to maintain protection. |
| Encryption in use | Compromised data in use allows access to encrypted data, whether at rest or in transit. For example, someone with access to random access memory might extract the encryption key from that memory. Once they have this key, they can decrypt the data that's stored in an encrypted state. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | During the authentication process, the system confirms the user's legitimacy as an authorized user. This often involves entering login credentials like a username and password to access specific system parts. Modern methods increasingly employ multi-level or two-factor authentication. |
| Authorization | A user's permissions within the system are defined by their authorization. This dictates if the user can read, create, delete, or modify database files. It can also influence a user's capability to add or delete users and files from the system. |
| Accounting | Accounting involves tracking a user's activities based on their system permissions. It logs which databases have been accessed, the actions taken during access, and the identity of the user who first entered the system. |

**\***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 |  |
| [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 |