**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 | Validate all input data to ensure it meets specified format, length, range, and data types that are expected for the program to execute as intended. Proper input validation should prevent well known attacks such as SQL injection and buffer overflow which enables bad actors to access or execute malicious code. In web applications, this validation should be done in client-side code and can additionally be validated on server side. |
| 1. Heed Compiler Warnings | Compiler warnings often signal vulnerabilities or inconsistencies in code that bad actors can exploit. Compiler warnings can also indicate the use of deprecated code which can lead to vulnerabilities. Developers should use the highest warning level available and modify code to eliminate these warnings. Static testing tools which scan libraries for known vulnerabilities should also be utilized to improve code security. |
| 1. Architect and Design for Security Policies | Developers should plan for the implementation and enforcement of security policies which align with organizational needs. The design should restrict components of the system to only the rights they need to carry out their jobs and nothing more. Doing so will prevent an attacker who has gained access to the system from elevating their privileges further. |
| 1. Keep It Simple | Software architecture should be kept as simple as possible to reduce the likelihood of introducing vulnerabilities into the system through the improper implementation, configuration, or application of the system. Doing so will also create a more manageable system which makes it easier to trace how an attacker infiltrated the system in the event of an attack. |
| 1. Default Deny | Systems should by default block all access and only permit access which meets specific conditions. |
| 1. Adhere to the Principle of Least Privilege | Components of the system should only be granted the permissions necessary to carry out their intended function. By doing so, developers reduce the possibility for attackers who have gained access to elevate privileges further. |
| 1. Sanitize Data Sent to Other Systems | Data sent to other subsystems, such as command prompts, relational databases, and vendor software should be sanitized properly prior to being sent. This prevents attackers from invoking additional functionality of these components such as injection attacks. This varies from input validation as the calling component understands the context in which the call was made, whereas subcomponents are simply invoked. |
| 1. Practice Defense in Depth | Implement multiple defense strategies at each system layer to reduce the likelihood of a vulnerability being exploitable. For instance, a system may implement secure coding techniques, proper network security configurations, and secure runtime environments to reduce the likelihood a vulnerability is exploitable by an attacker. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques should employ the use of identifying and eliminating known vulnerabilities through a variety of testing techniques. This can include static and dynamic testing of code, penetration testing, and reviewing code prior to being sent to production to prevent vulnerabilities from reaching the public. |
| 1. Adopt a Secure Coding Standard | A secure coding standard should be created and properly documented as early in the system lifecycle to reduce vulnerabilities and create an effective plan for the proper implementation of secure coding principles. This develops trust with stakeholders and reduces costs by reducing time spent later in development fixing security issues. |

### 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** | **Do not rely on user inputted data to be of the correct type.** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | All numerical input gathered from the user should first be retrieved as a string and converted to a number to ensure type safety. |

| **Noncompliant Code** |
| --- |
| The below code does not validate that the user inputted a number. If a user enters a string “hello” as input, the program can have unexpected behavior. |
| #include <iostream>  int main()  {  int num1, num2;  std::cout << "Enter the first number: ";  std::cin >> num1;  std::cout << "Enter the second number: ";  std::cin >> num2;  std::cout << num1 << " + " << num2 << " = " << num1 + num2;  return 0;  } |

| **Compliant Code** |
| --- |
| The below code first gathers the input from the user as a string and then attempts to cast it to an integer using the std::stoi function. If the casting throws an error, the invalid\_argument exception will be caught by the catch clause and an error message will be shown to the user before attempting to gather input from the user again. |
| #include <iostream>  #include <string>  void getInputInt(const std::string& prompt, int& var) {  std::string input;  while (1) {  std::cout << prompt;  std::cin >> input;  try {  var = std::stoi(input);  return;  }  catch (const std::invalid\_argument& e) {  std::cout << "Error: The value you entered is not a number." << std::endl;  }  }  }  int main()  {  int num1, num2;  getInputInt("Enter the first number: ", num1);  getInputInt("Enter the second number: ", num2);  std::cout << num1 << " + " << num2 << " = " << num1 + num2;  return 0;  } |

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

| **Principles(s):** Validate Input Data, Adopt a Secure Coding Standard  This standard maps to the Validate Input Data principle as it is important to validate the input received from users and ensure it is in a valid format and meets the correct specifications. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axios Bauhaus Suite | 7.2.0 | CertC-ERR34 |  |
| Clang | 3.9 | Cert-err34-c | Checked by clang-tidy |
| CodeSonar | 8.1p0 | BADFUNC.ATOF  BADFUNC.ATOI  BADFUNC.ATOL  BADFUNC.ATOLL  (customization) | Use of atof Use of atoi Use of atol Use of atoll  Users can add custom checks for uses of other undesirable conversion functions. |
| Parasoft C/C++ test | 2023.1 | CERT\_C-ERR34-a | The library functions atof, atoi and atol from library stdlib.h shall not be used |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Avoid overflowing integers when performing mathematical operations on numeric data types.** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Performing mathematical operations on large values of integers can result in integer overflow where the value exceeds the maximum value of the data type, resulting in a really low number and unexpected results. |

| **Noncompliant Code** |
| --- |
| The below code will result in integer overflow as there is no check to ensure that the mathematical operations performed on the variables does not exceed the maximum value that the variable data type can hold. |
| #include <iostream>  int main() {  int num1 = 2147483647; // largest value signed integer can hold  int num2 = 1;  std::cout << num1 << " + " << num2 << " = " << result; // result is -2147483648  } |

| **Compliant Code** |
| --- |
| The compliant code performs a check on the variables to ensure that it does not overflow the variables prior to performing the operations and throws an error if it does. |
| #include <iostream>  #include <limits>  template <typename T>  T add\_numbers(T const& num1, T const& num2) {  if (num1 > std::numeric\_limits<T>::max() - num2) {  throw std::overflow\_error();  }  return num1 + num2;  }  int main() {  int num1 = 2147483647; // largest value signed integer can hold  int num2 = 1;  try {  int result = num1 + num2;  std::cout << num1 << " + " << num2 << " = " << result;  }  catch (const std::overflow\_error& e) {  std::cout << e.what() << std::endl;  }  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard  Ensure that you prevent vulnerabilities like the above standard by using scanner tools to check for logical errors and follow a secure coding standard to ensure compliant code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | Integer-overflow | Fully checked |
| CodeSonar | 8.1p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  MISC.MEM.SIZE.ADDOFLOW  MISC.MEM.SIZE.BAD  MISC.MEM.SIZE.MULOFLOW  MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size  Integer overflow of allocation size  Multiplication overflow of allocation size  Subtraction underflow of allocation size  Addition overflow of size  Unreasonable size argument  Multiplication overflow of size  Subtraction underflow of size |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Helix QAC | 2024.1 | C2800, C2860  C++2800, C++2860  DF2801, DF2802, DF2803, DF2861, DF2862, DF2863 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Avoid Buffer Overflows by using std::string when gathering input** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | C strings can be non-memory safe and result in a buffer overflow if the destination buffer is not large enough to support the input buffer. Using std::string ensures that a buffer of adequate size is created before the value is copied into the destination buffer. |

| **Noncompliant Code** |
| --- |
| The code below uses a C-string of size ten. If the user enters a name larger than 10 characters, a buffer overflow will occur and unexpected behavior can occur including the execution of malicious code or access to other areas of memory. |
| [#include <iomanip>  #include <iostream>  int main()  {  char name[10];  std::cout << "Enter your name: ";  std::cin >> name;  std::cout << "Your name is " << name << std::endl;  } |

| **Compliant Code** |
| --- |
| The code below meets compliance by using std::string instead of a c-string since the >> operator on std::string is overloaded to read the size from the input stream and allocate enough memory before copying data into the buffer. This prevents buffer overflow from occurring. |
| #include <iomanip>  #include <iostream>  int main()  {  std::string name;  std::cout << "Enter your name: ";  std::cin >> name;  std::cout << "Your name is " << name << std::endl;  } |

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

| **Principles(s):** Validate Input Data, Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopting a Secure Coding Standard  Validating that the input received from the user has enough memory allocated follows validation of input data. By preventing known vulnerabilities that actors can exploit you are adopting a secure coding standard and adopting another layer of defense at the code level. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | Int-division-by-zero  Int-modulo-by-zero | Fully checked |
| CodeSonar | 8.1p0 | LANG.ARITH.DIVZERO  LANG.ARITH.FDIVZERO | Division by zero Float Division. By Zero |
| Coverity | 2017.07 | DIVIDE\_BY\_ZERO | Fully implemented |
| Cppcheck | 1.66 | zerodiv  zerodivcond | Context sensitive analysis of division by zero  Not detected for division by struct member / array element / pointer data that is 0  Detected when there is unsafe division by variable before/after test if variable is zero |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Avoid SQL injection by sanitizing and/or parameterizing SQL statements.** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Use of sanitization or parameterization of SQL statements reduces the possibility for SQL injection attacks by parsing the input for common SQL injection statements or by passing input as a string. |

| **Noncompliant Code** |
| --- |
| The below code is noncompliant because it executes a SQL statement without sanitization or parameterization. This can result in SQL injection be exploitable by an attacker. Attackers can easily inject SQL by entering their username as “’ AND 1=1 ’” |
| bool run\_query(sqlite3\* db, const std::string& username, const std::string& password, std::vector<UserRecord>& records)  {  // clear any prior results  records.clear();  char\* error\_message;  std::string sql = "SELECT \* FROM users WHERE username='" + username + "' AND password='" + password + "'";  // Execute SQL and stores result  int result = sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message);  // if SQL statement did not successfully execute display error  if (result != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

| **Compliant Code** |
| --- |
| The below code parameterizes SQL and sanitizes arguments to reduce the likelihood of a SQL injection attack. |
| bool hasSqlInject(const std::string& sql, char\*\* error\_message) {  // Regex pattern will match 'string'='string' or x=x  // where x is a number  std::regex pattern("('.\*'='.\*')|(([0-9]+)=([0-9]+))");  std::smatch matches;  // Detect if the sql query contains SQL injection  if (std::regex\_search(sql, matches, pattern)) {  std::string newErrorMessage = "Attempt at SQL injection found";  size\_t msgLen = newErrorMessage.length() + 1;  // Change pointer of error message to point to the newly created string  \*error\_message = new char[msgLen];  // Copy the contents of new error message into the pointer  strcpy\_s(\*error\_message, msgLen, newErrorMessage.c\_str());  return true;  }  return false;  }  bool run\_query(sqlite3\* db, const std::string& username, const std::string& password, std::vector<UserRecord>& records)  {  // clear any prior results  records.clear();  char\* error\_message;  std::string sql = "SELECT \* FROM users WHERE username='" + username + "' AND password='" + password + "'";  if (hasSqlInject(sql, &error\_message) ||  sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  } |

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

| **Principles(s):** Validate Input Data, Sanitize Data Sent to Other Systems, Practice Defense in Depth, Adopting a Secure Coding Standard  Preventing SQL injection requires validation of input received from the user and ensuring that even if an actor attempts to inject malicious code, that data is sanitized and does not reach other systems. This adds another layer of defense to protect systems that do not check the context of calling systems. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | IO.INJ.COMMAND  IO.INJ.FMT  IO.INJ.LDAP  IO.INJ.LIB  IO.INJ.SQL  IO.UT.LIB  IO.UT.PROC | Command injection  Format string injection  LDAP injection  Library injection  SQL injection  Untrusted Library Load  Untrusted Process Creation |
| Coverity | 6.5 | TAINTED\_STRING | Fully implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_C-STR02-a  CERT\_C-STR02-b  CERT\_C-STR02-c | Protect against command injection  Protect against file name injection  Protect against SQL injection |
| Polyspace Bug Finder | R2023b | CERT C: Rec. STR02-C | Checks for:  Execution of externally controlled command  Command executed from externally controlled path  Library loaded from externally controlled path  Rec. partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Avoid memory leakage by deallocating memory created using the new keyword** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Memory created using the new keyword is not automatically cleaned up because it is allocated on the heap instead of the stack. If you create an object using the new keyword, always ensure that you deallocate the memory using delete. |

| **Noncompliant Code** |
| --- |
| The below code is noncompliant because it dynamically allocates memory on the heap using the new keyword but does not deallocate memory once it is done. |
| #include <iostream>  int main() {  int\* arr = new int[10]; // Allocating memory on the heap  for (int i = 0; i < 10; ++i) {  arr[i] = i;  }  // Forgot to deallocate memory  return 0;  } |

| **Compliant Code** |
| --- |
| The below code is compliant because once the dynamically allocated memory is finished, the memory is freed using delete[] |
| #include <iostream>  int main() {  int\* arr = new int[10]; // Allocating memory on the heap  for (int i = 0; i < 10; ++i) {  arr[i] = i;  }    delete[] arr;  return 0;  } |

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

| **Principles(s):** Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding standard  Proper memory management ensures that memory is not exploitable by malicious actors and adds defense at the code level. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.1 | DF4761,  DF4762,  DF4766,  DF4767 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-MEM53-a | Do not invoke malloc/realloc for objects having constructors |
| Polyspace Bug Finder | R2023b | CERT C++: MEM53-CPP | Checks for objects allocated but not initialized (rule fully covered). |
| PVS-Studio | 7.29 | V630, V749 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Do not leave assertions in production code** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions should not be left in production code since once they are reached, the program terminates abruptly. If an assertion is left in production code, malicious actors can exploit this by creating a denial of service attack and gain an understanding of how the code works which can lead to further vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The below code is noncompliant because it uses assert to validate that the choice entered by the user is between 1 and 4 inclusive. This will result in an abrupt termination of the program if the user enters an invalid option rather than safely recovering. |
| #include <iostream>  #include <cassert>  void DisplayChoices() {  // Print out choices to user  }  int main()  {  int choice;  DisplayChoices();  std::cin >> choice;  assert(1 <= choice && choice <= 4);  std::cout << "Your choice is " << choice << std::endl;  } |

| **Compliant Code** |
| --- |
| The below code is compliant because it replaces the use of assert with conditional blocks which continuously prompt the user to enter a valid choice option rather than terminating the program abruptly if they entered an invalid choice option. |
| #include <iostream>  #include <cassert>  void DisplayChoices() {  // Print out choices to user  }  int main()  {  int choice;  while (1) {  DisplayChoices();  std::cin >> choice;  if (1 <= choice && choice <= 4) {  break;  }  else {  std::cout << "Invalid choice option" << std::endl;  }  }  std::cout << "Your choice is " << choice << std::endl;  } |

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

| **Principles(s):** Validate Input Data, Sanitize Data Sent to Other Systems, Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding standard  Removing assertions in code prevents an attacker from exploiting the system through a Denial of Service attack and thus provides an additional layer of defense at the code level. If you need to assert the values of data before being sent to other systems or that was provided by users, validate/sanitize the data and then provide a proper workflow that does not interrupt the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Coverity | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| Parasoft C/C++ test | 2023.1 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Avoid unhandled exceptions** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | If an exception is thrown, the exception should be safely handled, and safely recovered from when possible. Do not just leave an exception unhandled. |

| **Noncompliant Code** |
| --- |
| The below code attempts to gather input from the user, but abruptly terminates if the conversion from string to integer throws an exception. |
| #include <iostream>  #include <string>  void getInputInt(const std::string& prompt, int& var) {  std::string input;    std::cout << prompt;  std::cin >> input;  var = std::stoi(input);  }  int main()  {  int num1, num2;  getInputInt("Enter the first number: ", num1);  getInputInt("Enter the second number: ", num2);  std::cout << num1 << " + " << num2 << " = " << num1 + num2;  return 0;  } |

| **Compliant Code** |
| --- |
| In this example, if the input from the user is not a numeric value, an exception will be thrown and the exception is properly handled to safely recover and continue to gather input from the user until a valid value is provided. |
| #include <iostream>  #include <string>  void getInputInt(const std::string& prompt, int& var) {  std::string input;  while (1) {  std::cout << prompt;  std::cin >> input;  try {  var = std::stoi(input);  return;  }  catch (const std::invalid\_argument& e) {  std::cout << "Error: The value you entered is not a number." << std::endl;  }  }  }  int main()  {  int num1, num2;  getInputInt("Enter the first number: ", num1);  getInputInt("Enter the second number: ", num2);  std::cout << num1 << " + " << num2 << " = " << num1 + num2;  return 0;  } |

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

| **Principles(s):** Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding standard  Allowing a program to handle exceptions and recover gracefully follows a secure coding standard that does not interrupt the user experience and adds an additional layer of defense by not revealing underlying vulnerabilities in code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 |  |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable catch |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Close files when they are no longer needed** |
| --- | --- | --- |
| Input/Output | STD-008-CPP | Ensure all files are closed upon successful opening to prevent resource leakage or locking the file indefinitely. If a server opened up a file and never closes it, other components might not be able to access that file and memory and processing resources may still be allocated to the handling of that opened file. |

| **Noncompliant Code** |
| --- |
| The below code is noncompliant because after successful opening of the file with filename “myfile.txt” it never closes it even after it is done writing to it. This can result in the file being locked and other system components unable to access the file and resources still being allocated to keeping that file open. |
| #include <iostream>  #include <fstream>  int main() {  std::ofstream file("myfile.txt");  if (!file.is\_open()) {  std::cerr << "Failed to open file!" << std::endl;  return 1;  }  // Write some data to the file  file << "I'm writing some data!\n";  // File is not closed  return 0;  } |

| **Compliant Code** |
| --- |
| The below code is compliant because after successfully opening and writing to “myfile.txt” the file is closed, leaving it available for other components to access and deallocating resources to the file. |
| #include <iostream>  #include <fstream>  int main() {  std::ofstream file("myfile.txt");  if (!file.is\_open()) {  std::cerr << "Failed to open file!" << std::endl;  return 1;  }  // Write some data to the file  file << "I'm writing some data!\n";  file.close();  return 0;  } |

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

| **Principles(s):** Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding standard  Proper resource management is a part of adopting a secure coding standard to prevent leaking of resources which can lead to actors exploiting the vulnerability and causing a Denial-of-Service attack. Additionally, proper resource management ensures the user experience is not slowed down and reduces production costs. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |
| Klockwork | 2024.1 | RH.LEAK |  |
| Parsoft C/C++ test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Polyspace Bug Finder | R2023b | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Ensure enough space for null-terminator in c-strings** |
| --- | --- | --- |
| String Correctness | STD-009-CPP | C-strings require a null-terminator ‘\0’ which indicates the end of the string. If there is not enough room left for the null-terminator in the c-string, unexpected behavior and potential memory overflow can occur. |

| **Noncompliant Code** |
| --- |
| The code below is noncompliant as there is not enough room left for a null-terminator character to be added to the c-string. This can result in undefined behavior as the null-terminator indicates the end of the string. |
| #include <iostream>  #include <cstring>  int main() {  char str[5];  std::strcpy(str, "Hello");  std::cout << "String: " << str << std::endl; // Undefined behavior: str is not null-terminated  return 0;  } |

| **Compliant Code** |
| --- |
| The below code ensures compliance by leaving space for the null-terminator and adding it to the end of the string. This prevents unexpected behavior from happening. |
| #include <iostream>  #include <cstring>  int main() {  char str[6]; // Ensure enough space for null-termination  std::strcpy(str, "Hello");  str[5] = '\0'; // Ensure null-termination  std::cout << "String: " << str << std::endl; // Compliant: str is null-terminated  return 0;  } |

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

| **Principles(s):** Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding standard  Ensuring proper use of strings defends against actors looking to exploit memory through c-string vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 8.1p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |
| RuleChecker | 22.10 | Stream-input-char-array | Partially checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Ensure that division and remainder operations do not result in divide-by-zero errors** |
| --- | --- | --- |
| Data Value | STD-010-CPP | Diving by zero will result in an exception being thrown and can abruptly terminate the program which could result in a denial of service attack. Ensure proper validation of input if gathered by user. |

| **Noncompliant Code** |
| --- |
| The below code accepts values from the user and performs division on them but does not check if the value entered for the divisor is 0. This can lead to a divide by zero error being thrown and terminating the program abruptly. |
| #include <iostream>  int main() {  int dividend, divisor;  std::cout << "Enter the dividend: ";  std::cin >> dividend;  std::cout << "Enter the divisor: ";  std::cin >> divisor;  std::cout << dividend << " / " << divisor << " = " << dividend / divisor;  } |

| **Compliant Code** |
| --- |
| The below code meets compliance by properly checking if the user inputted divisor is zero before performing division on the values. The input is continued to gather until the user enters a number not equal to zero. |
| #include <iostream>  int main() {  int dividend, divisor;  std::cout << "Enter the dividend: ";  std::cin >> dividend;  while (1) {  std::cout << "Enter the divisor: ";  std::cin >> divisor;  if (divisor != 0) {  break;  }  else {  std::cout << "Error: cannot divide by zero" << std::endl;  }  }  std::cout << dividend << " / " << divisor << " = " << dividend / divisor;  } |

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

| **Principles(s):** Validate Input Data, Practice Defense in Depth, Use Effective Quality Assurance Techniques, Adopt a Secure Coding standard  All input data should be verified and validated before performing calculations to ensure there is not abrupt termination of the program which would interrupt the user experience. Basic quality assurance testing could verify this standard is being upheld. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | Int-division-by-zero  Int-modulo-by-zero | Fully checked |
| CodeSonar | 8.1p0 | LANG.ARITH.DIVZERO  LANG.ARITH.FDIVZERO | Division by zero Float Division. By Zero |
| Coverity | 2017.07 | DIVIDE\_BY\_ZERO | Fully implemented |
| Cppcheck | 1.66 | zerodiv  zerodivcond | Context sensitive analysis of division by zero  Not detected for division by struct member / array element / pointer data that is 0  Detected when there is unsafe division by variable before/after test if variable is zero |

### 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 will be used to create a secure pipeline for production that ensures code adheres to the standards defined in this document. Prior to reaching production, possible attack vectors and impact analysis will occur which will carry over into the secure design of the system. Test-driven design will drive the design process and secure builds will take place before reaching the verification and testing stage of the pipeline. The verification and testing phase ensures that the code complies with the standards defined in this document and that the possible attack vectors have been properly secured.

After verification and testing, the code will be deployed into production with the correct configurations to ensure defense against outside threats. Penetration testing will be performed to look for potential vulnerabilities that can be exploited. While in production, proper accounting of events will occur to monitor for unwarranted activity and warn against potential threats. In the event of a threat, a team will respond to the attack and utilize logs to isolate and secure the issue. Once the threat has been neutralized and production has been stabilized, a recall of events that took place will occur to determine the next course of action for development to prevent future security vulnerabilities.

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

### Create Policies for Encryption and Triple A

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | The practice of encryption at rest ensures that data that is not actively being used is secured from potential attackers if the data is stolen by keeping the data encrypted. By doing so, the attacker would need to break the encryption by getting a hold of the corresponding key to decrypt the data, or by brute forcing the encryption which would take an abundant amount of processing power and time that it would make the task unfeasible. In practice, data that is encrypted at rest has a corresponding encryption key and decryption key. |
| Encryption in flight | The practice of encryption in flight ensures that data being transferred from one system to another is secured against an attacker that can access it (i.e. man-in-the-middle attacks). To secure data in transit, a secure connection needs to be established using SSL/TLS between the systems. |
| Encryption in use | The practice of encryption in use ensures that data that is currently being used is secured from an attacker through authentication and authorization. Specifically, only those who SHOULD be able to access the data can access it once they have identified themselves via the proper credentials. This practice typically involves an Identity Access Management system. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This policy identifies users accessing the system by validating login credentials. Only users who can successfully login can access the system. This adds a layer of security to prevent anyone from accessing and making changes to the system |
| Authorization | This policy grants authenticated users permissions based on their role or user group. By default, users are granted only the necessary permissions to carry out their work and nothing more to prevent malicious actors from being able to escalate privileges in the event they infiltrate the system. |
| Accounting | This policy accounts for the monitoring of events regarding the system by logging events such as login attempts (successful/unsuccessful), changes to databases, files accessed with timestamps, or creation of new resources. This ensures that in the event a malicious actor can infiltrate the system, Quality Assurance can view the logs in the system to gain info about the attack. Additionally, it allows us to have lockdown procedures which prevent certain system behaviors in the event suspicious activity occurs based on log events. |

### Map the Principles

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

* Operating system logs (8, 9)
  + - Operating system logs adhere to the Practice of Defense in Depth by ensuring that in the event a vulnerability is exploited, other security measures are in place to leave a detailed trace of the events that took place to remediate the damage.
    - Operating system logs adhered to the Use of Effective Quality Assurance Techniques by providing events which can be used for system audits
* Firewall logs (5, 8, 9)
  + - Firewall Logs follow the Default Deny standard by denying those who should not be accessing the system and logging the information for accounting
    - Firewall logs adhere to the Practice of Defense in Depth by ensuring that in the event a vulnerability is exploited, other security measures are in place to leave a detailed trace of the events that took place to remediate the damage.
    - Firewall logs adhered to the Use of Effective Quality Assurance Techniques by providing events which can be used for system audits
* Anti-malware logs (5, 7, 8, 9)
  + - Anti malware logs follow the Default Deny standard by denying things that could be deemed suspicious to prevent an attack
    - Anti-malware logs follows Sanitization of Data sent to other systems by not trusting all files and ensuring that there is not malicious data that could be trickled down to subsystems
    - Anti-malware logs adhere to the Practice of Defense in Depth by ensuring that in the event a vulnerability is exploited, other security measures are in place to leave a detailed trace of the events that took place to remediate the damage.
    - Anti-malware logs adhered to the Use of Effective Quality Assurance Techniques by providing events which can be used for system audits

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 |  |
| 2.0 | 04/07/2024 | Completion of project | Jerrett Gonsalves |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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