**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 designed to prevent hackers from inserting bad code into a text input field. The program should check the user’s input to ensure that it is of correct type. For example, if the program expects a string and the user inputs a number the program should be designed to recognize the incorrect input. Also, the program should validate character length input to prevent overflow or underflow. |
| 1. Heed Compiler Warnings | This principle dictates the importance of programmers to not ignore compiler warnings as these warnings may identify some possible vulnerabilities within the code. Programmers should use the highest warning levels for the compiler to ensure that the majority of possible vulnerabilities are flagged. |
| 1. Architect and Design for Security Policies | A program’s architecture should be designed in a way that prioritizes security and access levels. Permissions to access certain functionality of a program should be restricted to ensure that only specified individuals like system administrators have the right privilege levels to make any major changes to the program, while giving users a separate level of access. |
| 1. Keep It Simple | Programmers should design code to be as simple as possible. When unnecessary complexity is added it not only slows down a program, but also makes it difficult to debug and update, and may also pose additional vulnerabilities that hackers can exploit. |
| 1. Default Deny | This principle dictates that users should have limited access level to the program. The program should prevent the user to access any functionality for which the user’s permission level was not granted to. |
| 1. Adhere to the Principle of Least Privilege | This principle dictates that users should have the minimum required permissions in order to be able to successfully complete their intended task and not much more than that. This prevents users from unintentionally or intentionally accessing areas of the program or making updates to the program when they were not authorized to do so. |
| 1. Sanitize Data Sent to Other Systems | When the program receives data, it should sanitize this data by checking the inputs to ensure that they do not pose a threat to other systems such as sending data to a relational database. |
| 1. Practice Defense in Depth | This principle dictates that a system should have multiple layers of defense against malicious users. These layers may consist of both software such as a firewall, an email scanner, and antivirus, all working together to ensure that if a virus is able to pass through one layer of security then there are additional layers that will be able to catch it. There are also physical layers that companies use such as fences, locked doors, security guards and surveillance cameras. |
| 1. Use Effective Quality Assurance Techniques | The use of quality assurance techniques such as penetration testing, code audits, integration testing, and external reviews ensure that potential program vulnerabilities are identified and resolved quickly and effectively. |
| 1. Adopt a Secure Coding Standard | It is important to adopt a secure coding standard for programmers to be consistent in the way they write software and how they optimize their defensive capabilities. This is also very important when you have a team of software developers and allows everyone to be on the same page and work effectively as a unit. |

### 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 | Data Type Coding Standard |

| **Noncompliant Code** |
| --- |
| The use of a data type such as a string to hold a type of data that it is not designed to contain such as a number. |
| String numPersons; |

| **Compliant Code** |
| --- |
| The correct data type is being utilized for the type of data that is being stored. |
| unsigned long numPersons; |

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

| **Principles(s):**   * **Principle 1** maps to this standard. Input data should be validated to ensure it is of the correct type. A hacker may try to insert malicious code or try to crash a program by entering an incorrect data type. * **Principle 7** maps to this standard. Data should be checked to ensure it is the correct type before being sent to other systems. * **Principle 8** maps to this standard. Validating data type of input data is one layer of defense that can be used in a defense-in-depth strategy. * **Principle 10** maps to this standard because it should be a standard practice of secure coding to declare variables with a type that makes sense for the data that the variable will hold. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | Type  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for type errors, such as signed integer overflow and float conversion overflow |
| CppCheck | 2.6 | String  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for the misuse of strings |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Data Value Coding Standard |

| **Noncompliant Code** |
| --- |
| The data type used is too small for the value the variable will hold. |
| char speedOfLight; |

| **Compliant Code** |
| --- |
| The data type used is of sufficient size for the value the variable will hold. |
| unsigned long long speedOfLight; |

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

| **Principles(s):**   * **Principle 1** maps to this standard because the size of the input data should be validated to prevent overflow/underflow conditions. * **Principle 7** maps to this standard because data should be analyzed to detect potential overflow or underflow conditions before this data is sent to other systems. * **Principle 8** maps to this standard because the detection and prevention of overflow/underflow scenarios is one layer of defense that can be used in a defense-in-depth strategy. * **Principle 10** maps to this standard because it should be standard practice of secure coding to declare variables with a maximum/minimum size that makes sense for the data they will contain. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | Bounds checking  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for out of bounds errors including buffer overflow error |
| CppCheck | 2.6 | Type  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for type errors, such as signed integer overflow and float conversion overflow |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | String Correctness Coding Standard |

| **Noncompliant Code** |
| --- |
| The first 20 characters of “this\_string” are copied to “chars”. If the string is more than 20 characters then not all string data will be copied. |
| char \*this\_string;  char chars[20];  strncpy(chars, this\_string, sizeof(chars)); |

| **Compliant Code** |
| --- |
| Before copying the string check the size to determine if the entire string can be copied to “chars” |
| char \*this\_string = NULL;  char chars[20];  if (this\_string == NULL) {  //handle null pointer  }  else if (strlen(this\_string) >= sizeof(chars)) {  //handle overly long string  }  else {  strcpy(chars, this\_string);  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Somewhat Likely | Low | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | Null pointer checks  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for null pointer dereferencing and undefined null pointer arithmetic |
| CppCheck | 2.6 | Bounds checking  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for out of bounds errors including partial string write leading to buffer that is not null terminated |
| CppCheck | 2.6 | String  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for misusage of strings |
| Clang Static Analyzer | 10.8 | cplusplus.StringChecker  <https://clang.llvm.org/docs/analyzer/checkers.html#cplusplus-stringchecker-c> | This tool checks std::string operations |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | SQL Injection Coding Standard |

| **Noncompliant Code** |
| --- |
| SQL query is run but there is no check to prevent SQL Injection attacks. |
| if (sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  //handle query failure  }  return true; |

| **Compliant Code** |
| --- |
| Check the SQL query to search for a “ or value=value” type of attack. If a potential attack is detected, return false. |
| if (sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  //handle query failure  }  std::size\_t found = sql.find(" or ");  if (found != std::string::npos) {  std::cout << "SQL Injection is blocked" << std::endl;  return false;  }  return true; |

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

| **Principles(s):**   * **Principle 1** maps to this standard because the input data should be validated to ensure that it does not contain malicious SQL code. * **Principle 7** maps to this standard because data should be analyzed to ensure it does not contain inserted SQL code before being sent to other systems. * **Principle 8** maps to this standard because preventing SQL injection attacks is one layer of defense that can be used in a defense-in-depth strategy. * **Principle 10** maps to this standard because the prevention of SQL injection attacks should be part of the standard practice when it comes to secure coding. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Highly | Low | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.2 Enterprise Edition | Injection flaw detection engine  <https://www.sonarqube.org/features/security/?gads_campaign=North-America-SonarQube&gads_ad_group=SonarQube-Security&gads_keyword=sonarqube%20vulnerability&gclid=Cj0KCQiA47GNBhDrARIsAKfZ2rC5WZklii3DFosA4VhIxzs0hOE9w01uvZFUsSdDp-JPiMRzQ3FvP6kaAlLUEALw_wcB> | This tool checks for code vulnerabilities including injection flaws |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Memory Protection Coding Standard |

| **Noncompliant Code** |
| --- |
| User input is read into the user\_input variable without the size first being checked. |
| const std::string username = “John Smith”;  char user\_input[30];  std::cout << "Enter a value: ";  std::cin >> user\_input; |

| **Compliant Code** |
| --- |
| User input is stored into a temporary string variable. The size of the temporary string is checked and if it is greater than or equal to the size of user\_input an error message is printed. Otherwise the contents of the temporary string are copied to user\_input. |
| const std::string username = “John Smith";  char user\_input[30];  std::cout << "Enter a value: ";  std::string temp\_string;  std::cin >> temp\_string;  while (temp\_string.length() >= 30) {  std::cout << "Error; buffer overflow blocked. Input must be fewer than 30 characters." << std::endl;  std::cout << "Enter a value: ";  std::cin >> temp\_string;  }    strncpy\_s(user\_input, temp\_string.c\_str(), sizeof(user\_input)); |

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

| **Principles(s):**   * **Principle 1** maps to this standard because the size of the input data should be validated to prevent overflow/underflow conditions. * **Principle 7** maps to this standard because data should be checked to detect potential overflow/underflow before being sent to other systems. * **Principle 8** maps to this standard because the detection and prevention of overflow/underflow scenarios is one layer of defense that can be used in a defense-in-depth strategy. * **Principle 10** maps to this standard because checking and prevention of possible overflow/underflow scenarios should be a standard practice of secure coding. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | Memory leaks (address not taken, class variables, function variables, struct members)  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for errors that may cause memory leaks |
| CppCheck | 2.6 | Bounds checking  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for out of bounds errors including array index out of bounds, pointer arithmetic overflow, buffer overflow, etc. |
| Clang Static Analyzer | 10.8 | cplusplus.NewDeleteLeaks  <https://clang.llvm.org/docs/analyzer/checkers.html#cplusplus-newdeleteleaks-c> | This tool checks for memory leaks |
| Clang Static Analyzer | 10.8 | cplusplus.InnerPointer  <https://clang.llvm.org/docs/analyzer/checkers.html#cplusplus-innerpointer-c> | This tool checks for inner pointers after memory re/deallocation |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions Coding Standard |

| **Noncompliant Code** |
| --- |
| Multiplication is performed without using an assertion to prevent overflow. |
| total\_size = size \* sizeof(char \*); |

| **Compliant Code** |
| --- |
| The assert statement checks to make sure that the multiplication will not cause overflow before the multiplication is performed. |
| assert(size <= SIZE\_MAX/sizeof(char \*));  total\_size = size \* sizeof(char \*); |

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

| **Principles(s):**   * **Principle 8** maps to this standard because testing code with assertions it is one layer of defense that can be used in a defense-in-depth strategy. * **Principle 9** maps to this standard because assertions are an effective quality assurance technique that can be used to identify code vulnerabilities. * **Principle 10** maps to this standard because it should be standard practice of secure coding to use assertions for testing purposes. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Somewhat likely | Low | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | Assert  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool gives a warning if assert statements have side effects |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Exceptions Coding Standard |

| **Noncompliant Code** |
| --- |
| “function\_throw” throws an exception but it is not caught by “the\_function” or “main”. |
| void function\_throw() noexcept(false);    void the\_function() {  function\_throw();  }    int main() {  the\_function();  } |

| **Compliant Code** |
| --- |
| A try-catch is used in “main” to handle the exception. |
| void function\_throw() noexcept(false);    void the\_function() {  function\_throw();  }    int main() {  try {  the\_function();  } catch (…) {  //handle error  }  } |

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

| **Principles(s):**   * **Principle 2** maps to this standard because if compiler gives a warning, it may indicate a particular line of code where an exception may occur. Therefore, the developer can use this information to know where they should account for any possible exception handling. * **Principle 8** maps to this standard because handling exceptions is one layer of defense that can be used in a defense-in-depth strategy. * **Principle 9** maps to this standard because exception handling is an effective quality assurance technique that can be used to prevent issues and resolve errors. * **Principle 10** maps to this standard because it should be standard practice of secure coding to handle all exceptions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | Exception Safety  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool provides various checks for exception safety (throwing exceptions in destructors or during invalid state, throwing exception in noexcept or nothrow(), etc.) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Operators | STD-008-CPP | Use of operators |

| **Noncompliant Code** |
| --- |
| A bitwise operator is incorrectly used when checking for equality. |
| if (numMen() & numWomen() == 0) {  /\*. . . \*/  } |

| **Compliant Code** |
| --- |
| A logical operator is used. |
| if (numMen () && numWomen () == 0) {  /\*. . . \*/  } |

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

| **Principles(s):**   * **Principle 9** maps to this standard because external review and source code audits are effective quality assurance techniques that can be used to identify logic errors. * **Principle 10** maps to this standard because it should be standard practice of secure coding to properly choose between bitwise operators and Boolean-like operands for a given situation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | Boolean  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for the boolean type errors including use of bool in a bitwise expression |
|  |  |  |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Functions | STD-009-CPP | Function calling |

| **Noncompliant Code** |
| --- |
| The main function calls multiplyNums with only one argument but multiplyNums requires two arguments. |
| int multiplyNums (int x, int y) {  int total = x \* y;  }  int main (void) {  int number1;  int number2;  cin >> number1;  cin >> number2;  multiplyNums(number2);  return 0;  } |
|  |

| **Compliant Code** |
| --- |
| The correct number of arguments are used when calling multiplyNums. |
| int multiplyNums (int x, int y) {  int total = x \* y;  }  int main (void) {  int number1;  int number2;  cin >> number1;  cin >> number2;  multiplyNums (number1, number2);  return 0;  } |

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

| **Principles(s):**   * **Principle 2** maps to this standard because a compiler will probably give an error warning in this scenario, therefore a developer should ensure that these lines of code are fixed. * **Principle 10** maps to this standard because it should be standard practice of secure coding to ensure all function calls use the correct number and type of arguments. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Somewhat likely | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | Check function usage  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for appropriate function usage including invalid input values, attempts to call discouraged function, etc. |
| Clang Static Analyzer | 10.8 | core.CallAndMessage  <https://clang.llvm.org/docs/analyzer/checkers.html#core-callandmessage-c-c-objc> | This tool checks for logical errors for function calls |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integer operations | [STD-010-CPP | Use of signed integer operations |

| **Noncompliant Code** |
| --- |
| Both integers are added together but the code does not check for overflow first. |
| signed int bird\_count;  signed int squirrel\_count;  cout << “How many birds?” << endl;  cin >> bird\_count;  cout << “How many squirrels?” << endl;  cin >> squirrel\_count;  signed int total\_creatures = bird\_count + squirrel\_count; |

| **Compliant Code** |
| --- |
| The if/else block first checks to see whether overflow will occur. The addition only happens if no overflow is detected. |
| signed int bird\_count;  signed int squirrel\_count;  cout << “How many birds?” << endl;  cin >> bird\_count;  cout << “How many squirrels?” << endl;  cin >> squirrel\_count;  if ((squirrel\_count > 0) && (bird\_count > (INT\_MAX - squirrel\_count))) ||  ((squirrel\_count < 0) && (bird\_count < (INT\_MIN - squirrel\_count)))){  //handle error  }  else {  signed int total\_creatures = bird\_count + squirrel\_count;  } |

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

| **Principles(s):**   * **Principle 1** maps to this standard because the input data should be validated for size to prevent an overflow/underflow scenario while running a mathematical calculation. * **Principle 7** maps to this standard because values should be validated to detect potential overflow/underflow before sending the result of the calculation to other systems. * **Principle 8** maps to this standard because the detection and prevention of overflow/underflow scenarios is one layer of defense that can be used in a defense-in-depth strategy. * **Principle 10** maps to this standard because checking and prevention of possible overflow/underflow scenarios should be a standard practice of secure coding. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Somewhat likely | Low | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | Bounds checking  <https://sourceforge.net/p/cppcheck/wiki/ListOfChecks/> | This tool checks for out of bounds errors including pointer arithmetic overflow |

### 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 have a major role in the Pre-Production Verify and Test phase for the current DevOps pipeline at Green Pace. The key role of automation will be to deliver a streamlined way to detect errors and any possible threats that the development team, reviewers, or the software program may have missed. The automation tools are to be utilized as one possible layer of defense in the DiD policy. It will act as a backup layer as the development and review teams will be responsible to ensure that any possible risks are alleviated while maintaining the software’s intended functionality. However, automation should not be relied upon as the primary layer of defense since its tools may miss some key errors that may cause unintended program behavior. Automation should be used as a tool to double check the program as changes are made throughout the production and development cycles.

### 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 | Low | High | 5 |
| STD-002-CPP | High | Likely | Low | High | 5 |
| STD-003-CPP | Low | Somewhat Likely | Low | Low | 3 |
| STD-004-CPP | High | Likely | Low | High | 5 |
| STD-005-CPP | High | Likely | Low | High | 5 |
| STD-006-CPP | Medium | Somewhat Likely | Low | Medium | 4 |
| STD-007-CPP | Medium-High | Likely | Medium | High | 5 |
| STD-008-CPP | Medium | Somewhat Likely | Low | Medium | 3 |
| STD-009-CPP | Low | Somewhat Likely | Low | Low | 1 |
| STD-0010-CPP | High | Somewhat Likely | Low | Medium | 4 |

### 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 in rest allows for data that is not being used to be secured. An example of encryption at rest is data that is being archived in a database. Encryption in rest is applied by the use of algorithms to guard the data in the company database. Employees who have the proper access privileges to the database will be able to view the unencrypted protected data. This type of policy will ensure that only authorized personnel are able to view the encrypted data. This is especially important in industries such as the financial and healthcare sectors. |
| Encryption at flight | Encryption in flight ensures the security of data that is being transmitted. This policy will detail how a company should send/retrieve data via HTTPS or SFTP file/data transfer protocols as a default measure. This will help to prevent unauthorized or malicious parties from intercepting the sensitive data while it is being sent or received. |
| Encryption in use | Encryption in use will safeguard the data that is in use on the company’s various software applications. The safeguard measures include encryption of specific data fields that may be confidential such as bank accounts or dates of birth. This policy will also drive the analysis of specific individuals who request access to these data fields in order to identify whether the request is legitimate or malicious in its intent. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication policy requests users to enter their assigned credential information at login to access the company system. This information can be username and password as well as 2-factor authentication. Green Pace employees will complete a series of trainings for each of the software applications that they will access before they can be added as a user. The policy will only allow access to users that have a legitimate business need to access the system and will require manager review and approval. Users will only access applications that will allow them to complete their work duties and will only have the minimum necessary permissions to the system. Every year their manager will review a list of applications that their direct reports have access to and will either approve or deny access. |
| Authorization | Authorization policy will require that employees can have access permissions to a given application at a minimum necessary level to be able to complete their daily duties. User roles will be segmented to allow some individuals greater access levels than others. For example, data architects and database administrators of a data warehouse will have greater access privilege than analysts. If an employee needs greater access, they must fill out a formal request which needs to be approved by their people manager and review by IT. This policy will limit the chances of inappropriate data access and will ensure that data is safeguarded. |
| Accounting | The accounting policy will ensure that there is a proper audit trail in place to track and analyze user behavior and access to the company’s systems. Audit trails will collect information on user access and any changes that are made to the system such as CRUD database actions. This policy will also track which specific information the user is viewing and can identify whether there was a legitimate need for the user to view that information. |

**\***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 |  |
| 2.0 | 04/17/2022 | Completed Policy | Alex Bevz | [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 |