**Green Pace Developer: Security Policy Guide Project**



# 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 | Ensure that any data inputted is handled to be valid. Discard any input that is invalid and do not enlist it in the program for execution. Be sure to return errors to the user should input be invalid, especially to guide them to valid input. |
| 1. Heed Compiler Warnings | Pay attention to any compiler warnings that populate during the development process. These are warnings generated by respective IDEs of innate warnings that exist in the code. These must be resolved for an application to run smoothly without errors during compilation. Compiler warnings also give hints to potential security issues or exploits. |
| 1. Architect and Design for Security Policies | Keep security policies in mind all throughout the architect and design processes of an application. Refer to security policies during development to ensure that the application is being developed securely. |
| 1. Keep It Simple | Keep application code as simple and straightforward as possible to reduce white noise throughout code. Ensure that vulnerabilities are excised with simple code. |
| 1. Default Deny | By default, deny access to all resources and functionalities. Only delegate access once permission is explicitly granted. Ensure that only authorized users or processes access system resources. |
| 1. Adhere to the Principle of Least Privilege | The users and system processes of any application should only be given the minimum level of access needed to perform their job functions. |
| 1. Sanitize Data Sent to Other Systems | Before sending any data to another system, clean the data thoroughly to ensure it is received well. It is also important to sanitize data to prevent data leakage and to comply with data protection regulations. |
| 1. Practice Defense in Depth | Add thorough layers of defense throughout the system. This ensures that if one layer falls through, other layers can compensate and keep the system secure should a vulnerability occur. |
| 1. Use Effective Quality Assurance Techniques | Be sure to be efficient when using quality assurance techniques to uphold the highest quality all throughout the application. Examples of effective quality assurance techniques are automated testing, code reviews, and security audits. |
| 1. Adopt a Secure Coding Standard | Detail a secure coding standard for your application and adhere to it end-to-end from design, development, to deployment. This is essential for protecting application stakeholders, their data, and trust in developers and application owners. |

### 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 | Ensure that the appropriate data type is used for data that is held to ensure proper data integrity and operations throughout the program. |

| **Noncompliant Code** |
| --- |
| This code erroneously assigns an empty string to a number declared as int. It also erroneously assigns an integer as a string data type. This is a mismatch. |
| int number = "";  string string = 0; |

| **Compliant Code** |
| --- |
| This code correctly assigns an integer number with a 0 numeric value, and a string as a string datatype initialized with an empty string. |
| int number = 0;  string string = ""; |

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

| **Principles(s):** a. 2: Heed Compiler Warnings: When a data type is inappropriately assigned, a compiler warning will appear. For our security policy, it is essential to ensure that data types are always appropriate for their purposes; thus, heed compiler warnings that point to incorrectly assigned types. Compiler warnings are often indicators of security and operational issues.  b. 10: Adopt a Secure Coding Standard: Appropriate data type usage is essential to secure coding standards.  c. 1: Validate Input Data: Ensuring appropriate data type usage is a key step in validating input data. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | TYPE.MISMATCH | TYPE.MISMATCH checks code areas where inappropriate data types are used through static code analysis. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Ensure arithmetic operations do not lead to overflow to prevent unexpected behavior on numbers. |

| **Noncompliant Code** |
| --- |
| This noncompliant code adds increment to result without checking if an overflow has occurred. It continues to add increment to result until the loop is over. |
| T add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  result += increment;    }  return result;  } |

| **Compliant Code** |
| --- |
| This compliant code adds a check to see if subtracting result from the max limit results in a value smaller than the increment. If it is smaller, then overflow occurs. Thus it throws an overflow error to be caught later. |
| T add\_numbers(T const& start, T const& increment, unsigned long int const& steps)  {  T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  if (std::numeric\_limits<T>::max() - result < increment) {  throw std::overflow\_error("Overflow! Adding increment leads to too large of a result.");  }  else {  result += increment;  }  // else return the result    }  return result;  } |

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

| **Principles(s):** a. 1: Validate Input Data: Validating input data will handle situations where arithmetic operations may lead to overflow and thus cause unexpected behavior.  b. 3: Architect and Design for Security Policies: Preventing overflow is essential for security policies that protect against common vulnerabilities in the design phase. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Low | P2 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | arithmeticOverflow | arithmeticOverflow automatically checks for any arithmetic overflow issues through arithmetic analysis. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Use valid references, pointers, and iterators to reference elements of a basic\_string |

| **Noncompliant Code** |
| --- |
| After calling replace(), data is actually invalid, so the usage of data in g is undefined behavior. |
| #include <iostream>  #include <string>    **extern** **void** g(**const** **char** \*);    **void** f(std::string &exampleString) {  **const** **char** \*data = exampleString.data();    // ...    exampleString.replace(0, 2, "bb");    // ...    g(data);  } |

| **Compliant Code** |
| --- |
| This code uses an algorithm to perform the replacement. This is preferred to coding your own solution for string replacement. |
| #include <algorithm>  #include <string>    **void** f(**const** std::string &input) {    std::string email{input};    std::replace(email.begin(), email.end(), ';', ' ');  } |

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

| **Principles(s):** a. 9: Use Effective Quality Assurance Techniques: Automated testing, code reviews, and security audits can verify that pointers and references are always valid, preventing null pointer dereferences or out-of-bounds errors.  b. 1: Validate Input Data: The appropriate references to elements of a basic\_string can be further handled with input validation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | CERT C++: STR52-CPP | [CERT C++: STR52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr52cpp.html) | Checks for use of invalid string iterator (rule partially covered). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection by validating the input SQL query and ensuring that no extra additions to the query can be made. |

| **Noncompliant Code** |
| --- |
| This noncompliant code does not check the provided SQL query against anything and allows any appendages to occur before the SQL query is ran. All records including passwords are returned as a result of the where condition being ignored due to the OR appendage. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  records.clear();  char\* error\_message;  if(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;  }  …  switch (rand() % 4)  {  case 1:  injectedSQL.append(" or 2=2;");  break;  case 2:  injectedSQL.append(" or 'hi'='hi';");  break;  case 3:  injectedSQL.append(" or 'hack'='hack';");  break;  case 0:  default:  injectedSQL.append(" or 1=1;");  break;  }  }    return run\_query(db, injectedSQL, records); |

| **Compliant Code** |
| --- |
| This compliant code adds an if condition to check if the SQL statement contains any or statements, effectively preventing SQL injection. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  if (sql.find("or") != std::string::npos) {  std::cout << "Injected SQL: " << sql << std::endl;  std::cerr << "SQL Injection detected. Cancelling query execution." << std::endl;  return false;  } |

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

| **Principles(s):** a. 1: Validate Input Data: This principle critically holds up this standard as implementing this principle directly prevents SQL injection.  b. 3: Architect and Design for Security Policies: SQL injection is a common vulnerability and must be protected against in security policies.  c. 8: Practice Defense in Depth: Handling SQL injections within code is one layer of defense in depth, which can be implemented with parameterized queries, input validation, and auditing. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SQLMap | 1.5.7 | Dynamic analysis | Automatically tests for SQL injection exploits |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Manage and clean memory properly to prevent opening vulnerabilities or causing crashes in the application. |

| **Noncompliant Code** |
| --- |
| Malloc() allocates an object, but it is not freed before text\_buffer’s pointer ends. |
| #include <stdlib.h>    **enum** { BUFFER\_SIZE = 32 };    **int** f(**void**) {  **char** \*text\_buffer = (**char** \*)**malloc**(BUFFER\_SIZE);  **if** (text\_buffer == NULL) {  **return** -1;    }  **return** 0;  } |

| **Compliant Code** |
| --- |
| In this compliant code, text\_buffer’s pointer is deallocated through the free() method. |
| #include <stdlib.h>    **enum** { BUFFER\_SIZE = 32 };    **int** f(**void**) {  **char** \*text\_buffer = (**char** \*)**malloc**(BUFFER\_SIZE);  **if** (text\_buffer == NULL) {  **return** -1;    }    **free**(text\_buffer);  **return** 0;  } |

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

| **Principles(s):** a. 2: Heed Compiler Warnings: Compiler and build warnings alert developers to possible memory leaks and memory issues.  b. 3: Architect and Design for Security Policies: Memory management is a core part of security policies.  c. 9: Use Effective Quality Assurance Techniques: Memory issues can also be detected with quality assurance techniques like stress testing. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P6 | L4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.17.0 | Memcheck | Automatically detects memory errors |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-LLL] | Use assertions to validate code early in the development process. Also use assertions to cover checks for occurrences that may not be readily apparent to yourself or other developers. They serve as a type of coding flag to ensure that all developers are on the same page regarding the function and results of the code. |

| **Noncompliant Code** |
| --- |
| The following code does not use an assertion but rather relies on a comment, which is unclear and the values are left to be unknown. |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | **int** i = 0;  **int** j = 9;    **for**( ; (i < 10) ; (++i, --j) ) {      array1[i] = array2[j];  }    // at this point x should be 0 | |

| **Compliant Code** |
| --- |
| The code asserts that after the for loop operation, j is equal to -1 due to the nature of the j decrementing within the loop. It serves as a clarity checkpoint in code that may be unclear. |
| |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | **int** i = 0;  **int** j = 9;    **for**( ; (i < 10) ; (++i, --j) ) {      array1[i] = array2[j];  }    ASSERT( j == -1 ); | |

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

| **Principles(s):** a. 9: Use Effective Quality Assurance Techniques: Automated testing can notify developers of areas in code where assertions are not present.  b. 10: Adopt a Secure Coding Standard: Enforcing consistent assertions is a secure coding standard that contributes to the security of an application |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.6 | assertUsage | Identifies areas in code where asserts are not used |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Ensure that thrown exceptions are caught properly so that code issues are handled and validated. |

| **Noncompliant Code** |
| --- |
| This noncompliant code properly throws an error when the max condition is met, but does not implement a try catch block to catch the exception. |
| void MyFunc(int c)  {  if (c > numeric\_limits< char> ::max())  {  throw invalid\_argument("MyFunc argument too large.");  }  //...  }  int main()  {  //...  return 0;  } |

| **Compliant Code** |
| --- |
| This code properly throws an exception, and when the function is called with an argument that is too large, it properly catches the exception thrown within the function definition. It then outputs an error with the exception details and returns a -1 to stop the program and signifiy an error. |
| void MyFunc(int c)  {  if (c > numeric\_limits< char> ::max())  {  throw invalid\_argument("MyFunc argument too large.");  }  //...  }  int main()  {  try  {  MyFunc(256); //cause an exception to throw  }  catch (invalid\_argument& e)  {  cerr << e.what() << endl;  return -1;  }  //...  return 0;  } |

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

| **Principles(s):** a. 2: Heed Compiler Warnings: Compilers will notify when catch statements do not accompany try statements, indicating the absence of a throw statement somewhere in the code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 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** | **Name of Standard** |
| --- | --- | --- |
| Data Integrity | [STD-008-CPP] | Preserve the integrity of data by ensuring that inputs do not lead to buffer overflow, preventing the exposure or modification of other parts of the code. |

| **Noncompliant Code** |
| --- |
| This code implements a buffer via a char array. It takes in user input and sets the character limit to 20 characters. If the user input exceeds 20 characters, buffer overflow occurs and account\_number is modified. |
| const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::cout << "Enter a value: ";  std::cin >> user\_input;  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl; |

| **Compliant Code** |
| --- |
| This code implements the string class to store user\_input as a string. User input can now be of any length and account\_number is left unmodified and safe. Buffer overflow is effectively prevented. |
| std::cout << "Buffer Overflow Example" << std::endl;  const std::string account\_number = "CharlieBrown42"; // account\_number must be right before input  std::string user\_input; // alternative to character array  std::cout << "Enter a value: ";  std::cin >> user\_input; // collect the number from the user  std::cout << "You entered: " << user\_input << std::endl; // display the number  std::cout << "Account Number = " << account\_number << std::endl; // display the account\_number |

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

| **Principles(s):** a. 1: Validate Input Data: Validating input data allows buffer overflow cases to be handled and prevented.  b. 5: Default Deny: Sometimes, buffer overflow can lead to unauthorized access to resources. Actively prevent buffer overflow to default deny such access. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P5 | L4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity Static Analysis | 2021.07 | BUFFER\_SIZE | Detects potential buffer overflows through static analysis |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Value | [STD-009-CPP] | Ensure arithmetic operations do not lead to underflow, preventing unexpected behavior on numbers throughout the program. |

| **Noncompliant Code** |
| --- |
| This code continuously decrements from result without proper checks or error throwing when underflow occurs. Since there is no comparison to the minimum numeric limit of the data type, underflow will occur and data values of numbers will be compromised. |
| T subtract\_numbers(T const& start, T const& decrement, unsigned long int const& steps)  {  T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  result -= decrement;  }  return result;  } |

| **Compliant Code** |
| --- |
| This code performs the subtract number operation, but during the decrementing process adds additional checks to see if subtracting the decrement from the current result will reach below the numeric limit minimum of the given data type. If it reaches below the minimum after decrementing, then an underflow error is thrown correctly. |
| T subtract\_numbers(T const& start, T const& decrement, unsigned long int const& steps)  {  T result = start;  for (unsigned long int i = 0; i < steps; ++i)  {  if (  result < decrement  ||  result - decrement > result  ||  (decrement > 0 && result < std::numeric\_limits<T>::min() + decrement)  )  {  // The above conditions for underflow are met so throw an underflow error exception  throw std::underflow\_error("Underflow! Subtracting decrement leads result below the minimum limit.");  }  else {  result -= decrement;  }  }  return result;  } |

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

| **Principles(s):** • 1: Validate Input Data: Validating input data will handle situations where arithmetic operations may lead to underflow and thus cause unexpected behavior.  • 3: Architect and Design for Security Policies: Preventing underflow is essential for security policies. The standard considers security policies since handling underflow prevents unexpected behavior. It is essential to mitigate such risks preemptively. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Highly Probable | Medium | P5 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 5.3 | ARITH.UNDERFLOW | Analyzes arithmetic in code and detects potential underflow calculations |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Management | [STD-010-CPP] | Do not access memory that is already freed, to prevent unexpected behavior from occurring. |

| **Noncompliant Code** |
| --- |
| The c\_str method is being called on a string object. However, once the string object is destroyed at the end of the assignment, the pointer from c\_str will point to freed memory. Thus, when any elements of that pointer are accessed, undefined behavior occurs. |
| #include <string>    std::string str\_func();  void display\_string(const char \*);    void f() {  const char \*str = str\_func().c\_str();  display\_string(str); /\* Undefined behavior \*/  } |

| **Compliant Code** |
| --- |
| Here, a local copy \*cstr is created from calling the c\_str method on the string object. The string str is thus valid when display\_string is called on it, without accessing released memory. |
| #include <string>    std::string str\_func();  **void** display\_string(**const** **char** \*s);    **void** f() {    std::string str = str\_func();  **const** **char** \*cstr = str.c\_str();    display\_string(cstr);  /\* ok \*/  } |

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

| **Principles(s):** a. 2: Heed Compiler Warnings: Compilers will notify the developer when freed memory is accessed, so the standard builds on this principle by encouraging developers to handle such cases.  b. 3: Architect and Design for Security Policies: Preventing unexpected behavior from occurring in applications is a main mission for security policies, especially as a preemptive measure. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Highly Probable | High | P7 | L5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5.0 | USE\_AFTER\_FREE | Detects specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

### 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 for enforcing and complying to standards should be implemented throughout the pre-production pipeline, namely throughout the build and verify/test landscapes. Automation is an iterative process through the pre-production development sequence. This is where code is mainly developed, and code should be frequently automated to ensure that development is aligning to coding standards. It is essential to catch such errors and vulnerabilities early before production rollouts. Because the DevSecOps pipeline is a cyclical process, automation can be iterated when production rolls into the pre-production process to enhance future production rollouts. Implementing automation in pre-production and iterating upon it will allow Green Pace to deliver secure, reliable, and high-quality software.*

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

### 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 at rest | Encryption at rest involves encryption to protect data on physical devices or in cloud environments. The goal is to protect data when it is not actively being used. Encryption at rest should be used to protect against attackers who gain physical access to storage devices. It allows protection through the usage of encryption keys. Utilize technologies such as BitLocker or FileVault to provide encryption at rest to protect storage devices for data not actively used. |
| Encryption in flight | This involves protecting data as it is transmitting between systems or networks. It protects against attacker interception. Examples are HTTPS and SSL/TLS. Data is encrypted as it is sent across networks. Always use secure protocols when transmitting data between systems to ensure that attackers do not intercept potentially vulnerable data. |
| Encryption in use | Outside of at rest and during transit, data is actively being used and processed. Encryption in use protects this data. Use a policy that ensures data remains encrypted even during computation to protect against unauthorized access during processing. Attackers may access data in use through code injection or memory scraping, so it is important to use practices that secure data during processing like homomorphic encryption and secure enclaves. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies the identity of the user to allow access to resources in a system. This is commonly done through user logins with credentials. Authentication also involves the addition of new users to an existing login database, where records are added to check if incoming users can be authenticated against the database. This applies for Green Pace’s client entrepreneurs to login and authenticate to the software systems. Authentication should always be utilized to ensure the proper users are tracked and accessing the correct system. |
| Authorization | Authorization ensures that the user is authorized to do respective actions. This involves checks for if users are authorized to access certain files. This also determines a user’s level of access at different points throughout the system. Entrepreneur users must be authorized and given the least privileges to perform their required functions within the Green Pace software for utmost security. Users will only access what they are permitted to access. Green Pace can implement role-based access control for authorization. |
| Accounting | Accounting is an auditing process that tracks user activities and related security data in a system. This can involve time tracking (when users login and logout of the system), resource monitoring, and log management (can involve user activity like files that are accessed by users, or addition of new users). It is critical for Green Pace to implement accounting to ensure compliance and optimal security preparation. |

**\***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.] | 06/14/2024 | Final Policy | Marissa Sihapanya |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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