**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 | Any data coming from an untrusted source needs to be validated. Untrusted sources include external applications and user input. Validation methods include parameterization, whitelisting, bounds checking, and numeric overflow/underflow. |
| 1. Heed Compiler Warnings | Compiler warnings need to be addressed and resolved and not ignored. While an application may run successfully on the source machine’s architecture, it may not run on different architectures. |
| 1. Architect and Design for Security Policies | As much as is possible, portable code should be used. This requires avoiding compiler-specific instructions. This document also is used to create consistent code use and set expectations. |
| 1. Keep It Simple | Code should comply with modern syntax practices to keep code neat and readable. The simplest form of code should be used to achieve an objective without sacrificing security practices. |
| 1. Default Deny | As mentioned, whitelisting should be used. Rather than attempt to predict and disallow every issue, it is better to list what is allowed and block all else. |
| 1. Adhere to the Principle of Least Privilege | All users (including admins) should only have as much access as is needed, and only for as long as it is needed. In this way, if an account is compromised, the damage will be minimized and controlled. |
| 1. Sanitize Data Sent to Other Systems | All data being sent should be sanitized and parameterized. For example, user input should be sanitized before sending queries to an SQL database application. In addition, programmers should be aware of and familiar with what data is required and how it should be organized before sending it to another system. |
| 1. Practice Defense in Depth | Multiple layers of defense covering every entry is necessary to mitigate damage from intruders. For example, an email filter should be applied in combination with giving users least privilege. The email filter may not stop all attacks from being successful, but reduced privileges might be successful in such an event. Firewalls, antivirus, and user education should also be included. |
| 1. Use Effective Quality Assurance Techniques | Programmers should test their own code as they write it. Test cases and vulnerability scans should also be used. Thorough testing and quick responses to found insecurities are necessary. It may be beneficial depending on budget to preform penetration testing from hired white hat hackers. |
| 1. Adopt a Secure Coding Standard | Also as mentioned, this document provides the secure coding standards that are to be upheld within Green Pace. It provides organization and consistency across development teams and sets expectations upfront. |

### 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 | Use only C++ built-in data types or data types provided in the standard library. |

| **Noncompliant Code** |
| --- |
| Creating a new type “counter” to use to track amounts. |
| typedef <int> <counter> // creating custom type “counter”  typedef counter<int> numPoints = 0;  numPoints++; |

| **Compliant Code** |
| --- |
| Using built-in “int” type and standard library “string” type. |
| std::string player = “John”; // std::string from standard library  int points = 0; // int from built in type  points++; |

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

| **Principles(s):** #4: Keep It Simple; #10 Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Medium | High | Medium | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Ensure that values are kept within minimum and maximum type limits to prevent numeric overflow and underflow. |

| **Noncompliant Code** |
| --- |
| add() function has no numeric overflow or underflow checking. |
| int add(const int userNum1, const int userNum2) {  return userNum1 + userNum2; // numeric overflow!  }  void main() {  const int userNum1 = INT\_MAX;  const int userNum2 = 1;  int sum = add(userNum1, userNum2);  return 0;  } |

| **Compliant Code** |
| --- |
| add() function includes numeric overflow prevention. |
| int add(const int userNum1, const int userNum2) {  int sum;  // Overflow check  if (INT\_MAX – userNum2 < userNum1) {  sum = userNum1 + userNum2;  }  else {  // numeric overflow prevented!  }  return sum;  }  void main() {  const int userNum1 = INT\_MAX;  const int userNum2 = 1;  int sum = add(userNum1, userNum2); // returns error  return 0; // program completes successfully  } |

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

| **Principles(s):** #1 Validate Input Data; #2 Heed Compiler Warnings; #8 Practice Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | When using char arrays (as opposed to std::string), use strncat\_s() to concatenate arrays and strncpy\_s() to copy strings. |

| **Noncompliant Code** |
| --- |
| Insecure methods to concatenate and copy char arrays. This does not perform bounds checking on userName and causes a buffer overflow. |
| char greetingTemplate[] = "hello ";  char userName[] = "johnathan";  char greeting[11];  strcpy\_s(greeting, greetingTemplate); // cannot specify copy length  strcat\_s(greeting, userName); // buffer overflow! |

| **Compliant Code** |
| --- |
| Proper secure use of strncpy\_s() and strncat\_s(). “johnathan” is shortened to “john”. |
| char greetingTemplate[] = "hello ";  char userName[] = "johnathan";  char greeting[11];  const int MAX\_LENGTH = 4;  strncpy\_s(  greeting,  sizeof(greeting),  greetingTemplate,  sizeof(greetingTemplate) // can control copy length  );  strncat\_s(greeting,  sizeof(greeting),  userName,  MAX\_LENGTH // enforce copy length; prevents overflow  ); |

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

| **Principles(s):** #1 Validate Input Data; #2 Heed Compiler Warnings; #3 Architect and Design for Security Policies; #10 Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Sanitize all user input using parameterization before passing SQL query to database. |

| **Noncompliant Code** |
| --- |
| Concatenates query with user input without sanitization. |
| std::string query = “SELECT \* FROM USERS WHERE NAME=”;  std::string username;  cin >> username;  query.append(username); |

| **Compliant Code** |
| --- |
| Combines query with user input safely using parameterization. |
| std::string query = “SELECT \* FROM USERS WHERE NAME=?1 AND PASSWORD=?2”;  std::string username;  std::string password;  sqlite3\_stmt \* stmt;  std::cin >> username;  std::cin >> password;  password = hashPassword(password);  if (sqlite3\_prepare\_v2(db, query.c\_str(), query.length(), &stmt, nullptr) != SQLITE\_OK) {  // handle error  }  sqlite3\_bind\_text(stmt, 1, username.c\_str(), username.length(), nullptr);  sqlite3\_bind\_text(stmt, 2, password.c\_str(), password.length(), nullptr);  sqlite3\_step(stmt); |

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

| **Principles(s):** #1 Validate Input Data; #3 Architect and Design for Security Policies; #7 Sanitize Data Sent to Other Systems |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Ensure arrays and pointers have proper bounding and have been initialized when inserting data into it. |

| **Noncompliant Code** |
| --- |
| Buffer is not initialized properly, attempts to be accessed, loop goes beyond bounds, and pointer arithmetic is off. |
| char buffer[27];  char\* character;  for (int i = 0; i <= sizeof(buffer); i++) { // loop runs 1 beyond buffer  // bounds (buffer overflow and  // overwrites null terminator)  std::cout << buffer[i] << std::endl; // attempt to print  // uninitialized memory  }  buffer[2] = 'a';  char\* myPointer = buffer + 3; // incorrect pointer arithmetic  std::cout << \*myPointer << std::endl; // prints uninitialized memory |

| **Compliant Code** |
| --- |
| Buffer is initialized properly, loop is bounded appropriately, and pointer arithmetic is valid. |
| char buffer[27];  char\* character;  char letter = ‘a’;  for (int i = 0; i < sizeof(buffer) - 1; i++) { // loop bounded correctly  buffer[i] = letter;  std::cout << buffer[i] << std::endl; // prints initialized memory  letter++;  }  char\* myPointer = buffer + 3; // correct pointer arithmetic; prints ‘d’  std::cout << \*myPointer << std::endl; // prints successfully |

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

| **Principles(s):** #1 Validate Input Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Use assertions when creating and using test cases. |

| **Noncompliant Code** |
| --- |
| Testing add() function without use of assert(). Code is messy and returns less detail (i.e. line error numbers) |
| int add(const int num1, const int num2) {  return num1 + num2;  }  void testAddFunction() {  if (add(12, 34) != 46) {  std::cout << “Error! Unexpected result in add() function\n”;  }  if (add(73, 29) != 73 + 29) {  std::cout << “Error! Unexpected result in add() function\n”;  }  } |

| **Compliant Code** |
| --- |
| Makes use of assert to alert error and show detail and keep code concise. |
| #include <assert.h>  int add(const int num1, const int num2) {  return num1 + num2;  }  void testAddFunction() {  assert(add(12, 34) == 46);  assert(add(73, 29) == 73 + 29);  } |

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

| **Principles(s):** #1 Validate Input Data; #2 Architect and Design for Security Policies; #8 Practice Defense in Depth; #9 Use Effective Quality Assurance Techniques; #10 Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |
| GoogleTest | 1.12.1 | GoogleTest | Code verifier for unit tests |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Use try-catch exceptions when unexpected errors arise and not for user input. |

| **Noncompliant Code** |
| --- |
| Using exceptions for user input is not considered an unexpected error. |
| #include <string>  try {  const int MAX\_LENGTH = 10;  std::string name;  cin >> name;  if (name.length() > MAX\_LENGTH) {  throw std::invalid\_argument(“Input exceeds max length.”);  }  }  catch (std::invalid\_argument e) {  std::cout << e.what() << std::endl;  } |

| **Compliant Code** |
| --- |
| Opening a file is expected to be successful. |
| #include <fstream>  try {  fstream fs;  fs.open(“file.txt”);  }  catch (std::system\_error e) {  std::cout << e.what() << std::endl;  }  Catch (…) {  std::cout << “Unknown error” << std::endl;  } |

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

| **Principles(s):** #3 Architect and Design for Security Policies; #8 Practice Defense in Depth; #9 Use Effective Quality Assurance Techniques; #10 Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Function return values | STD-008-CPP | Always verify status return codes when a function or method allows. |

| **Noncompliant Code** |
| --- |
| Doesn’t catch and verify returned error code. |
| bool add(static int num1, static int num2, int& sum) {  if (INT\_MAX – num1 < num2) {  sum = num1 + num2;  }  else {  return false;  }  return true;  }  void main() {  userNum1 = 5;  userNum2 = 9;  userSum;  add(userNum1, userNum2, userSum); // ignores returned value with  // status result  cout << userSum << std::endl;  } |

| **Compliant Code** |
| --- |
| Catches and verifies returned error code. |
| bool add(static int num1, static int num2, int& sum) {  if (INT\_MAX – num1 < num2) {  sum = num1 + num2;  }  else {  return false;  }  return true;  }  void main() {  bool result = false;  userNum1 = 5;  userNum2 = 9;  userSum;  result = add(userNum1, userNum2, userSum);  **if (!result) {**  **// Error**  **}**  } |

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

| **Principles(s):** #3 Architect and Design for Security Policies; #8 Practice Defense in Depth; #9 Use Effective Quality Assurance Techniques #10 Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Array Memory Access | STD-009-CPP | Use [] operator to access char arrays and .at() to access std::string instead of using pointer arithmetic. |

| **Noncompliant Code** |
| --- |
| Changing array elements with pointer arithmetic and [] for std::string. Valid syntax, but more prone to errors. |
| int grades[] = { 82, 94, 76 };  const int EXTRA\_CREDIT = 5;  std::string grats = "Good job.";  int\* index;  index = grades + 1;  \*index += 5;  grats[8] = '!';  for (int g : grades) {  if (g >= 90) {  std::cout << grats << std::endl;  }  } |

| **Compliant Code** |
| --- |
| Changing array elements with [] for char array and .at() for std::string which is more secure. |
| int grades[] = { 82, 94, 76 };  const int EXTRA\_CREDIT = 5;  std::string grats = "Good job.";  grades[2] = grades[2] + EXTRA\_CREDIT;  grats.at(8) = '!';  for (int g : grades) {  if (g >= 90) {  std::cout << grats << std::endl;  }  } |

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

| **Principles(s):** #3 Architect and Design for Security Policies; #4 Keep it Simple; #8 Practice Defense in Depth; #9 Use Effective Quality Assurance Techniques; #10 Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Storage | STD-010-CPP | Ensure that plain passwords that were saved in memory are overwritten with 0s. |

| **Noncompliant Code** |
| --- |
| Encrypts password without clearing out plain text password memory |
| #include <string>  std::string password;  std::string encryptedPassword;  std::cin >> password;  encryptedPassword = encryptPassword(password);  return; |

| **Compliant Code** |
| --- |
| Erases memory after using plain password. |
| #include <string>  std::string password;  std::string encryptedPassword;  std::cin >> password;  encryptedPassword = encryptPassword(password);  for (int i = 0; i < password.size(); i++) {  password.at(i) = ‘0’;  }  return; |

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

| **Principles(s):** #3 Architect and Design for Security Policies; #8 Practice Defense in Depth; #9 Use Effective Quality Assurance Techniques; #10 Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Studio Enterprise | 17.4.33122.133 | VS Analyzer | Standard IDE for development |
| Cppcheck | 2.9 | Cppcheck | Static analysis for code checking |

### 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.

Visual Studio IDE includes an on-the-fly basic code analyzer to catch many basic errors as the developer writes code. Developers should be frequently compiling and testing their code every few lines which is where the built in analyzer will flag some issues. Once a particular section of code is complete (such as an entire function, method, or class), Cppcheck will need to be used for a deeper scan of the code. Issues flagged must be resolved or else be a false positive. Once flagged issues are resolved, the scan must be run again to verify that no new vulnerabilities have been introduced. This process should be repeated until no issues are visible or are all false positives.

Once a full block of code has been written (again, a function, method, or class), it must be tested using GoogleTest to ensure it functions as expected and handles unexpected use appropriately. Use of the written code and mock tests need to be conducted to ensure the code integrates with the rest of the system as well.

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

### 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 | Data at rest is data stored in non-volatile memory that is not being used. For example, data at rest would include files stored on a hard drive (HHD, SSD, USB) or flash drive (USB). Sensitive data at rest such as user data (username, password, bank information) needs to be encrypted should it be compromised and stolen by hackers. |
| Encryption at flight | Data at flight includes data that is actively being transferred such as over a network (intranet or internet). For example, a user who inserts their username and password on a server web page and attempts to login sends that data over the internet which could be sniffed by hackers. This data needs to be encrypted so that hackers are unable to understand the data they capture. |
| Encryption in use | Data in use is data stored in volatile memory such as RAM. For example, an opened file such as a word document is using a copy from the hard disk and storing it in active memory. This copy of data needs to be encrypted should hackers have a foothold and are looking at active memory. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | An authentication policy would include a system that verifies that the person attempting to access anything requiring permissions actually is who they say they are. |
| Authorization | Authorization policies should include ensuring that any given employee cannot preform a task without proper permissions and should only have the least amount of permissions necessary at any given time to do their work. |
| Accounting | An accounting policy could include logging security related events such as a user logging in or database query events including a timestamp. |

**\***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

| **STD-001-CPP:**  #4: Keep It Simple – Only using standard library to keep consistency between developers  #10 Adopt a Secure Coding Standard – Adopting broad standard to keep consistency |
| --- |
| **STD-002-CPP:**  #1 Validate Input Data – Ensure data is within bounds to prevent hacking  #2 Heed Compiler Warnings – Pay attention to compiler’s warnings about conversion data loss  #8 Practice Defense in Depth – Don’t default to defined behavior of wrapping. Don’t let it happen! |
| **STD-003-CPP:**  #1 Validate Input Data – Ensure data passed to built in methods are of the correct size and bounds  #2 Heed Compiler Warnings – Compiler should warn of using deprecated functions  #3 Architect and Design for Security Policies – Good standard being applied across all developers  #10 Adopt a Secure Coding Standard - Good standard being applied across all developers |
| **STD-004-CPP:**  #1 Validate Input Data - Never trust SQL queries due to SQL injection. Always use parameterization  #3 Architect and Design for Security Policies - Good standard being applied across all developers  #7 Sanitize Data Sent to Other Systems – SQL queries are processed in code and sent to SQL Database |
| **STD-005-CPP:**  #1 Validate Input Data – Don’t trust data passed by method caller such as bounds size |
| **STD-006-CPP:**  #1 Validate Input Data – Use assert to validate results based on passed input  #2 Architect and Design for Security Policies – Testing is part of the standard development process  #8 Practice Defense in Depth – Don’t assume code that works one way works all ways. Try to break it.  #9 Use Effective Quality Assurance Techniques - Testing is part of the standard development process  #10 Adopt a Secure Coding Standard - Testing is part of the standard development process |
| **STD-007-CPP:**  #3 Architect and Design for Security Policies – Standardized QA for coding  #8 Practice Defense in Depth - Don’t assume system calls are successful. Stop error from propagating  #9 Use Effective Quality Assurance Techniques - Standardized QA for coding  #10 Adopt a Secure Coding Standard - Standardized QA for coding |
| **STD-008-CPP:**  #3 Architect and Design for Security Policies – Make it a habit to verify status when offered by function  #8 Practice Defense in Depth – Calls ought to work but don’t assume they do.  #9 Use Effective Quality Assurance Techniques – Checking error codes makes for smoother code  #10 Adopt a Secure Coding Standard – Consistency being applied |
| **STD-009-CPP:**  #3 Architect and Design for Security Policies – Use more secure methods as a standard  #4 Keep it Simple – its easier, safer, and clearer code to use these operators  #8 Practice Defense in Depth – using other operators is valid syntax but more dangerous.  #9 Use Effective Quality Assurance Techniques - Consistency being applied |

| **STD-010-CPP:**  #3 Architect and Design for Security Policies – Designing for safe practices.  #8 Practice Defense in Depth – Passwords should not be able to be accessed, but erase just in case.  #9 Use Effective Quality Assurance Techniques – Safer and consistent using this standard.  #10 Adopt a Secure Coding Standard - Safer and consistent using this standard. |
| --- |

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 11/13/2022 | Added Coding Standards with examples. | Jason Verrill | [Insert text.] |
| 1.2 | 12/01/2022 | Added Risk Assessments | Jason Verrill | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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