**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 | For any data that the client is permitted to send, properly validate it to ensure it does not have bad intentions. Ensure that it does not exceed maximum length nor does it have any code that is not permissible. |
| 1. Heed Compiler Warnings | Listen to the warnings your compiler gives you, and do not ignore them. Correct the code rather than ignoring or suppressing the warnings where applicable. |
| 1. Architect and Design for Security Policies | Structure user access to ensure they are not granted more access than they need. Data is often confidential, so you must properly store it by encryption or other safeguarding means. |
| 1. Keep It Simple | Keep your design simple so it is easier to narrow down and correct errors, and colleagues will be able to quickly learn and navigate the code. |
| 1. Default Deny | Unless a user has a certain permission/privilege giving them access to certain content, deny them access. |
| 1. Adhere to the Principle of Least Privilege | Stick to minimal privileges required to complete a job, and when you are on elevated permissions, minimize the execution time to reduce chances of an attacker’s code being executed at a higher level. |
| 1. Sanitize Data Sent to Other Systems | Properly sanitize the data before passing it off somewhere else, such as an SQL database. Ensure it cannot unintentionally execute commands or inject code/data. |
| 1. Practice Defense in Depth | Properly set up multiple layers of defense in an effective and efficient manner so if one fails, there are still more standing. Do noy create too many layers since you’ll run into time complexity issues and will have a hard time tracking down an annoying bug. |
| 1. Use Effective Quality Assurance Techniques | Properly test your systems to ensure its integrity. Do not be afraid of allowing public or outside testers as they can offer a new perspective that you may not have noticed since you made the code to act one way. |
| 1. Adopt a Secure Coding Standard | Create a secure coding standard for your project. |

### 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 you are passing matching datatypes to functions |

| **Noncompliant Code** |
| --- |
| The below code will error at compile time since I am passing an integer where it is expecting a string |
| #include <iostream>  void hello(std::string name) {  std::cout << "Hello " << name << std::endl;  }  int main()  {  hello(5);  return 0;  } |

| **Compliant Code** |
| --- |
| This properly passes a string |
| #include <iostream>  void hello(std::string name) {  std::cout << "Hello " << name << std::endl;  }  int main()  {  hello("5");  return 0;  } |

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

| **Principles(s):** “Verify Proper Datatypes”. Since you have to check in your code that you are matching the datatype. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2021b | CERT C++: EXP58-CPP | Checks for incorrect data types for second argument of va\_start (rule fully covered) |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Give an appropriate value respective to the variable’s datatype |

| **Noncompliant Code** |
| --- |
| This will print out 1 instead of 5. |
| #include <iostream>  int main()  {  bool ID = true;  ID = 5;  std::cout << ID;  return 0;  } |

| **Compliant Code** |
| --- |
| This will properly print out 5. |
| #include <iostream>  int main()  {  int ID = 0;  ID = 5;  std::cout << ID;  return 0;  } |

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

| **Principles(s):** Validate variable values. Used to ensure ex: a boolean is assigned a boolean, not a number. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Ensure there is sufficient space when accepting input from the client to go into a char array representing a string. |

| **Noncompliant Code** |
| --- |
| This will cause a buffer overflow if the client inputs more than 5 characters. |
| #include <iostream>  int main()  {  char input[5];  std::cin >> input;  } |

| **Compliant Code** |
| --- |
| This will truncate the input after 5 characters. More can be done to instead tell the client they entered too many characters. |
| #include <iostream>  #include <iomanip>  int main()  {  char input[5];  std::cin >> std::setw(5) >> input;  } |

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

| **Principles(s):** Avoid Buffer Overflow in Arrays. Self-explanatory. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | BADFUNC.BO.\*  LANG.MEM.BO | A collection of warning classes that report uses of library functions prone to internal buffer overflows.  Buffer Overrun |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Properly sanitize input before directly passing it into an SQL statement |

| **Noncompliant Code** |
| --- |
| This code will allow OR statements to be injected |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  // clear any prior results  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;  } |

| **Compliant Code** |
| --- |
| This will detect if there is an “OR” statement which would manipulate the SQL |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  // clear any prior results  records.clear();  // Check for SQL injection for the 'OR' case provided  size\_t found1 = sql.find("or"); // If "or" / "OR" is found in the sql "string"  size\_t found2 = sql.find("OR"); // Figured this way was more as efficient as lowering a string since they both practically visit each character in the string, so might as well have 2 variables  if (found1 != std::string::npos || found2 != std::string::npos) { // If 'or' or 'OR' were found in the sql "string", terminate the query  std::cout << "SQL Injection OR encountered, terminating query. Please validate SQL." << std::endl;  return false;  }  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;  } |

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

| **Principles(s):** Sanitize SQL input data. Help prevent unintended operations or other injection. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2021.2 | CERT\_C-STR02-a  CERT\_C-STR02-b CERT\_C-STR02-c | Protect against command injection  Protect against file name injection  Protect against SQL injection |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly alert the user when a buffer overflow is going to occur |

| **Noncompliant Code** |
| --- |
| This code does not check the length of the user’s input and will cause a buffer overflow |
| #include <iomanip>  #include <iostream>  int main()  {  std::cout << "Buffer Overflow Example" << std::endl;  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 will check the length of the user’s input and warn the user of them exceeding the character limit, then terminate the program |
| #include <iomanip> // Will use this for std::setw  #include <iostream>  int main()  {  std::cout << "Buffer Overflow Example" << std::endl;  const std::string account\_number = "CharlieBrown42";  std::string initial\_input;  char user\_input[20];  std::cout << "Enter a value: ";  std::cin >> std::setw(21) >> initial\_input; // Put their input into initial\_input while setting a maximum width of 21 characters (since obviously anything after 20 is the overflow)  if (initial\_input.length() > 20) { // If the string exceeds 20 characters, it's an overflow  std::cout << "You have exceeded the 20 character limit. Terminating the program.." << std::endl;  return false;  }  strcpy\_s(user\_input, 20, initial\_input.c\_str()); // Copy the string onto the char array  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl;  } |

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

| **Principles(s):** Alert User for Buffer Overflow. Will prompt the user rather than terminating without explanation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | BADFUNC.BO.\*  LANG.MEM.BO | A collection of warning classes that report uses of library functions prone to internal buffer overflows.  Buffer Overrun |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Run proper unit tests on your code before shipping to production |

| **Noncompliant Code** |
| --- |
| This code will not check if a “Bob” is amongst us. |
| #include <iostream>  #include <cassert>  int main()  {  std::string name;  std::cin >> name;  std::cout << "Hi " << name << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| This code will throw an error if you enter “Bob”. No Bob’s allowed. |
| #include <iostream>  #include <cassert>  int main()  {  std::string name;  std::cin >> name;  assert(name != "Bob"); // we don't like Bob  std::cout << "Hi " << name << std::endl;  return 0;  } |

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

| **Principles(s):** Run Unit Tests. Help to ensure the code runs smoothly. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Your code must properly handle exceptions by using try and catch. |

| **Noncompliant Code** |
| --- |
| This code will exit with an error since you cannot divide by 0. |
| #include <iostream>  int main()  {  int first\_num = 5;  int second\_num = 0;  double result = 0;  result = first\_num / second\_num;  std::cout << "Result: " << result << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| This will output “An error has occurred.” And you can choose how to carry on after that. |
| #include <iostream>  int main()  {  int first\_num = 5;  int second\_num = 0;  double result = 0;  try {  if (second\_num == 0) {  throw "Cannot divide by 0";  }  result = first\_num / second\_num;  std::cout << "Result: " << result << std::endl;  }  catch (...) {  std::cout << "An error has occurred." << std::endl;  }  return 0;  } |

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

| **Principles(s):** Handle all exceptions. If there’s a will, there’s a way so don’t risk running risky code without proper protection. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Main-function-catchall  Early-catch-all | Partially Checked |
| Parasoft C/C++test | 2021.2 | 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 |
| Polyspace Bug Finder | R2021b | CERT C++: ERR51-CPP | Checks for unhandled exceptions |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Variable Naming** | [STD-008-CPP] | Do not set variable names that share the same name as a data type |

| **Noncompliant Code** |
| --- |
| This will error at compile time since it acknowledges you can’t have a variable name that’s a data type’s name. |
| int main()  {  int int = 5;  return 0;  } |

| **Compliant Code** |
| --- |
| This will run properly as the name of the variable is not a data type |
| int main()  {  int number = 5;  return 0;  } |

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

| **Principles(s):** Variable Names not being a data type. Self-explanatory. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2021.2 | CERT\_C-DCL01-a | Identifier declared in local or function prototype scope shall not hide an identifier declared in a global or namespace scope |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Proper Naming Convention** | [STD-009-CPP] | Properly name your variables so they accurately represent the data they’re representing. |

| **Noncompliant Code** |
| --- |
| This will confuse another programmer since 5 is clearly not a boolean, and its vagueness doesn’t make it clear what it’s representing. |
| int main()  {  int booleanValue = 5;  return 0;  } |

| **Compliant Code** |
| --- |
| This accurately describes what the number can represent without being misleading. |
| int main()  {  int appleAmnt = 5;  return 0;  } |

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

| **Principles(s):** Visually distinctive identifiers. Will help prevent random variable naming. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | LANG.ID.AMBIG | Typographically ambiguous identifiers |
| Parasoft C/C++ test | 2021.2 | CERT\_C-DCL02-a | Use visually distinct identifiers |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Wasteful Variables** | [STD-010-CPP] | Do not create and define a variable that will not serve a purpose later in the code. This takes up unnecessary memory and can confuse other programmers. |

| **Noncompliant Code** |
| --- |
| This does not use the defined variable, therefore using unnecessary memory. |
| #include <iostream>  int main()  {  int appleAmnt = 5;  std::cout << "You need to purchase more apples! You can never have enough!" << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| This uses all declared variables. |
| #include <iostream>  int main()  {  int appleAmnt = 5;  std::cout << "You have " << appleAmnt << " total apples!" << std::endl;  return 0;  } |

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

| **Principles(s):** Remove unused variables. Prevent memory from being wasted needlessly and clean up code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2021a | CERT C: Rec. MSC13-C | Checks for: Unused parameter, write without a further read |
| CodeSonar | 6.2p0 | LANG.STRUCT.UUVAL | Unused value |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation should take place during the pre-production phase. If they are in place before work begins, it can help ensure you maintain and follow the standards accordingly. It will tie in with the actual production phase to continue to ensure the code’s integrity.

### 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 | Medium | 2 | 3 |
| STD-002-CPP | Low | Unlikely | High | 1 | 3 |
| STD-003-CPP | High | Likely | Medium | 18 | 1 |
| STD-004-CPP | High | Likely | Medium | 18 | 1 |
| STD-005-CPP | High | Likely | Medium | 18 | 1 |
| STD-006-CPP | Low | Probable | High | 2 | 3 |
| STD-007-CPP | Low | Probable | Medium | 4 | 3 |
| STD-008-CPP | Low | Unlikely | Medium | 2 | 3 |
| STD-009-CPP | Low | Unlikely | Medium | 2 | 3 |
| STD-010-CPP | Low | Unlikely | Medium | 2 | 3 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | When data is on the disk, it should be encrypted. This prevents an attacker breaching the system and being able to read the data. |
| Encryption at flight | While data is being transmitted, it should be encrypted. You should use the most up-to-date protocols when transmitting data so attackers are unable to steal it while it is exposed/unencrypted. |
| Encryption in use | Protecting data that is in use. It’s best to do this by restricting who can access the data, and authorizing they’re privileged to access it. This prevents unauthorized users from retrieving the data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Verifies who the person is. Typically used with a username/password combination, and additional multi-factor authentication on top of that ranging from SSO systems to fingerprints. It applies to reaffirm who the user is, making it harder for someone to be impersonating them. |
| Authorization | This will check what the user is permitted to access. Typically enforced by different user level permissions and blocking certain content if the user does not meet the requirement. This is used to prevent an existing user/employee from reaching data they aren’t supposed to. |
| Accounting | This will keep logs of who access what and when they do it. It is useful after a breach to isolate which account may have been responsible. Outside of security, it’s useful for general statistics that you could benefit from such as hourly usage. |

**\***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 | 2/14/2022 | Completed Policy | Shawn Edouard | [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 |