**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 | The purpose of input validation is to verify that only proper data enters the information system, and any improper or malicious data is prevented. |
| 1. Heed Compiler Warnings | These warnings exist to deliver notification of potential error or issues inside of code. Errors prevent the compiling of code. Warnings do not prevent the code from compiling, but both of these are important for considering code modifications to prevent potential vulnerabilities. |
| 1. Architect and Design for Security Policies | We must consider the software architecture and design when we implement security policies. Separating systems into layers, or sub-systems, allows the developer to limit different types of users authorization or privileges based on their clearance level. For example, bank tellers do no need access to the whole system to perform their job function. |
| 1. Keep It Simple | Keeping a system small and simple reduces the risk of errors in the code, and while the software is in use. Minimizing the complexity of a system helps keep maintenance time down and reduces the difficulty. |
| 1. Default Deny | Access is denied by default and only allowed through the required conditions of the protection strategy. |
| 1. Adhere to the Principle of Least Privilege | Processes in the system should be able to execute with minimal required privileges needed to complete the task. Any elevated privileges should be invoked as little as possible and in as short a time-span as possible to reduce the chances that an attacker will have to execute code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Data sanitization checks data for potential issues before passing it along to other systems. Unused functions and calls that are made out context may pass and cause damage to the system such as SQL injection. |
| 1. Practice Defense in Depth | DiD, or defense in depth, says to have multiple layers of defense in a system. If one layer is penetrated then the others are still able to defend the system. Firewalls and input validation are great examples of this. |
| 1. Use Effective Quality Assurance Techniques | The use of testing and code audits are effective measures in a QA program. Internal and external security reviews help identify and correct vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Apply a coding standard in the language and platform to build strong security from the beginning. |
|  |  |

### 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] | We must obey the one-definition rule. |

| **Noncompliant Code** |
| --- |
| Two separate translation units define classes of the same name with definitions that are different. |
| // a.pp struct S {int a;}; // b.cpp class S {public int a;}; |

| **Compliant Code** |
| --- |
| Using a header file to introduce object in both translation units |
| // S.h struct S {int a;}; // a.cpp #include “S.h” // b.cpp #include “S.h” |

**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 – 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 | - |
| LDRA Tool Suite | 9.7.1 | 286 S, 287 S | Full Implementation |
| Astree | 20.10 | Undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading Type-compatibility, Definition-duplicate | Partial check |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP\_DCL60-a | Class, union, or enum name will be a unique identifier |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Don’t read uninitialized memory |

| **Noncompliant Code** |
| --- |
| An uninitialized local variable is evaluated in an expression to print its value. This results in undefined behavior. |
| #include <iostream>  void function()  {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The object is initialized before trying to print it. |
| #include <iostream>  void function()  {  int i = 0;  std::cout << i;  } |

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

| **Principles(s):** 1: Validate Input Data – 4: Keep It Simple – 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA Tool Suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partial implementation |
| Polyspace Bug Finder | R2021a | CERT C++: EXP53-CPP | This one checks for non-initialized variables and pointers. |
| Helix QAC | 2021.2 | C++2726, C++2727, C++2728, C++2961, C++2962, C++2963, C++2966, C++2967, C++2968, C++2971, C++2972, C++2973, C++2976, C++2977, C++2978 | - |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Don’t attempt creating an std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| The std::string object gets created by the result of the function call to std::getenv(). This function returns a null pointer upon failure and can lead to undefined behavior when an environment variable does not exist. |
| #include <cstdlib>  #include <string>  Void function() {  Std::string tmp(std::getenv(“TMP”));  If (!tmp.empty()) {  // …  }  } |

| **Compliant Code** |
| --- |
| Results returned by the call to std::getenv() get checked for a null pointer. |
| #include <cstdlib>  #include <string>  Void function() {  Const char \*tmpPtrVal = std::getenv(“TMP”);  Std::string tmp(tmpPtrVal ? tmpPtrVal : “”);  If (!tmp.empty()) {  // …  }  } |

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

| **Principles(s):** 2: Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klocwork | 2021.1 | NPD.CHECK.CALL.MIGHT NPD.CHECK.CALL.MUST NPD.CHECK.MIGHT NPD.CHECK.MUST NPD.CONST.CALL NPD.CONST.DEREF NPD.FUNC.CALL.MIGHT NPD.FUNC.CALL.MUST NPD.FUNC.MIGHT NPD.FUNC.MUST NPD.GEN.CALL.MIGHT NPD.GEN.CALL.MUST NPD.GEN.MIGHT NPD.GEN.MUST RNPD.CALL RNPD.DEREF | - |
| Helix QAC | 2021.2 | C++4770, C++4771, C++4772, C++4773, C++4774 | - |
| Astree | 20.10 | Assert\_failure | - |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-STR51-a | Avoid dereferencing null pointers |

#### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| If there are no precautions, then the untrusted data may maliciously change the query. |
| uName = getRequestString(“username”);  uPass = getRequestString(“password”);  sql = “SELECT \* FROM Users WHERE Name = + uName + “AND Pass = “ +uPass +“ |

| **Compliant Code** |
| --- |
| Sanitation and validation are the primary prevention measures of SQL injection. These are typically implemented in parameterized queries and stored procedures. |
| PreparedStatement pStmt = PreparedStatement();  Std::cin >> username;  Std::cin >> password;  sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;”, username, password}; |

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

| **Principles(s):** 1: Validate Input Data - 7: Sanitize Data Sent to Other Systems - 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| Parasoft Jtest | 2021.1 | CERT.IDS00.TDSQL | SQL injection protection |
| Coverity | 7.5 | SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_ FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Fortify | 1.0 | HTTP\_Response\_Splitting SQL\_Injection\_\_Persistence SQL\_Injection | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Don’t access the freed memory. |

| **Noncompliant Code** |
| --- |
| The variable is dereferenced after being deallocated. If the result of this results in a write-after-free event, then it can be exploited by an attacker to run arbitrary code with the privileges of the vulnerable process. |
| #include <new> struct S { void functOne(); };  void FunctTwo() noexcept(false) {  S \*s = new S;  delete s; // ...  s->funct();  } |

| **Compliant Code** |
| --- |
| Dynamic memory is not deallocated until it is no longer needed. |
| #include <new> struct S { void functOne(); };  void functTwo() noexcept(false) {  S \*s = new S; // ...  s->functOne();  delete s;  } |

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

| **Principles(s):** 2: Heed Compiler Warnings - 5: Default Deny - 6: Adhere to the Principle of Least Privilege - 9: Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Insure++ | - | - | Runtime error detection |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-MEM50-a | Don’t attempt to use freed resources |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Detects specific instances that memory is deallocated multiple times or read/written to the address of a freed pointer |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by Clang-Tidy but does not catch every violation of this rule |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Use static assertions to test the value of constant expressions |

| **Noncompliant Code** |
| --- |
| Using the assert() macro to assert a property concerning memory-mapped structure that is required for the code to run correctly. |
| #include <assert.h>  Struct timer {  Unsigned char MODE;  Unsigned int DATA;  Unsigned int COUNT; };  Int func(void) {  Assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| Constant expressions can use a preprocessing conditional statement can be used. |
| struct timer {  unsigned **char** MODE;  unsigned **int** DATA;  unsigned **int** COUNT; };  #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding" #endif |

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

| **Principles(s):**  2: Heed Compiler Warnings - 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 44 S | Full Implementation |
| ÉCLAIR | 1.2 | CC2.DCL03 | Full Implementation |
| Clang | 3.9 | Misc-static-assert | Clang-tidy |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | - |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Don’t terminate the program abruptly. |

| **Noncompliant Code** |
| --- |
| Here, a call to f(), registered as an exit handler with std::at\_exit(), might result in a call to std::terminate() because the throwing\_func() function might throw back an exception. |
| #include <cstdlib>  void throwing\_func() noexcept(false);  void f() { // Not invoked by the program except as an exit handler.  throwing\_func();  }  **int** main() {  if (0 != std::**atexit**(f)) {  // Handle error  } // ...  } |

| **Compliant Code** |
| --- |
| This handles all exceptions thrown back by throwing\_func() with f() and the exceptions are not rethrown |
| #include <cstdlib>  void throwing\_func() noexcept(false);  void f() { // Not invoked by the program except as an exit handler.  try {  throwing\_func();  }  catch (...) {  // Handle error  }  }  **int** main() {  if (0 != std::**atexit**(f)) {  // Handle error  } // ...  } |

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

| **Principles(s):** 9: Use Effective Quality Assurance Techniques - 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2021a | CERT C++: ERR50-CPP | Evaluates code for any implicit call to terminate(), partially covers the rule |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| CodeSonar | 6.1p0 | BADFUNC.ABORT BADFUNC.EXIT | Use of abort Use of exit |
| Klocwork | 2021.1 | MISRA.CATCH.ALL CERT.ERR.ABRUPT\_TERM | - |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-009-CPP] | Be sure to use valid iterating ranges. |

| **Noncompliant Code** |
| --- |
| In this scenario, std::for\_each() evaluates the first iterator with the second for equality after it has incremented the first. This is out of order and the first iterator will reach the past-the-end element of the range and result in an undefined behavior. |
| #include <iostream>  #include <vector>  #include <algorithm>  Void func(const std::vector <int> &c)  {  Std::for\_each(c.end(), c.begin(), [](int i))  {  Std::cout << i;  {  } |

| **Compliant Code** |
| --- |
| The iterator values are properly ordered and passed to std::for\_each() |
| #include <vector>  #include <algorithm>  #include <iostream>  void func(const std::vector<**int**> &c)  {  std::for\_each(c.begin(), c.end(), [](**int** i))  {  std::cout << i;  }  } |

**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 - 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.14 | V539, V662, V789 | - |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-CTR53-a CERT\_CPP-CTR53-b | Don’t use an iterator range that is not a real range or compare iterators from different containers |
| Astree | 20.10 | overflow\_upon\_dereference | - |
| Helix QAC | 2021.2 | C++ 3802 | - |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-009-CPP] | Write constructors that initialize variables, or objects, in the proper order (canonical). |

| **Noncompliant Code** |
| --- |
| This initializer tries to initialize someValue and dependsOnSomeValue in the wrong order. The declaration order of the variables does not follow the same pattern as initialization. When the system attempts to read the value of someValue, an unspecified value is stored in dependsOnSomeValue. |
| Class A  {  Int dependsOnSomeValue;  Int someValue;  Public A(int val) : someValue(val), dependsOnSomeValue(someValue+1) {}  }; |

| **Compliant Code** |
| --- |
| The declaration order of the variables is changed. This way the dependency can be ordered properly in the constructor for initialization. |
| class A  {  int someValue;  int dependsOnSomeValue;  public: A(int val) : someValue(val), dependsOnSomeValue(someVal + 1) {}  }; |

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

| **Principles(s):** 4: Keep it Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-OOP53-a | When initializing, list the items in the order in which they are declared |
| LDRA Tool Suite | 9.7.1 | 206 S | Full Implementation |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP53 | - |
| Astree | 20.10 | initializer-list-order | Full check |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-010-CPP] | Don’t access objects outside of their life cycle. |

| **Noncompliant Code** |
| --- |
| Here a pointer that points to an object is used to call a function before the beginning of the pointer’s life cycle. This results in an undefined behavior. |
| struct S  {  void func();  };  void Func() {  S \*s;  s->func();  } |

| **Compliant Code** |
| --- |
| The pointer has appropriate storage before the call to the function |
| struct S  {  void func();  };  void Func()  {  S \*s = new S;  s->func();  delete s;  } |

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

| **Principles(s):** 2: Heed Compiler Warnings - 10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-EXP54-a CERT\_CPP-EXP54-b CERT\_CPP-EXP54-c | Do not use freed resources, The address of an object with automatic storage shall not be returned from a function, The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist |
| CodeSonar | 6.1p0 | IO.UAC ALLOC.UAF | Use after close Use after free |
| Clang | 3.9 | -Wdangling-initializer-list | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |
| Astrée | 20.10 | return-reference-local dangling\_pointer\_use | Checked Partially |

### 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.

[Insert your written explanations here.]

### 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-002-CPP | High | Probable | Medium | High (12) | 1 |
| STD-003-CPP | High | Likely | Medium | High (18) | 1 |
| STD-004-CPP | High | Probable | Medium | High (12) | 1 |
| STD-005-CPP | High | Likely | Medium | High (18) | 1 |
| STD-010-CPP | High | Probable | High | Medium (6) | 2 |
| STD-006-CLG | Low | Unlikely | High | Low (1) | 2 |
| STD-007-CPP | Low | Probable | Medium | Low (4) | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | Low (4) | 3 |
| STD-001-CPP | High | Unlikely | High | Low (3) | 3 |

### 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 | This protects stored data in devices like hard drives, cell phones, tablets, and even in the cloud. Encryption prevents stolen data from being useful to an attacker unless they have the key. Industry regulations mandate this to protect sensitive data. |
| Encryption in flight | This is data in transit between devices in a network or across the internet. Data is encrypted to prevent theft during transmission. Things like email encryption, firewalls, and authentication help provide this defense. |
| Encryption in use | Encryption for data in use is called in-memory or runtime encryption. Similar to the other forms of data encryption, this method encrypts data with a key. Attackers cannot translate this data without the key. This prevents data loss or theft. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Confirms the identity of the user. This can be done with username and password, MFA, biometrics, and more. Credentials help the system ensure the user is who they say they are. |
| Authorization | This specifies the access rights of the user. Each user is given the minimum amount of privilege to each user type. This method provides limitation to users preventing unauthorized actions and attackers from stealing a low level set of credentials to access the entire system. |
| Accounting | Tracking and timestamping the actions of users and errors. This includes accessed resources, data transfers, and timestamping. This creates a trails of bread crumbs for someone to follow in case of a breach. |

**\***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 |  |
| 1.1 | 2/04/2025 | Mod 3 Milestone | Wesley Blackwell |  |
| 1.2 | 2/16/25 | Mod 6 Project 1 | Wesley Blackwell | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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