**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 process of filtering and checking incoming data from users or other sources. The goal is to prevent injection attacks by non-accepting any suspicious inputs. |
| 1. Heed Compiler Warnings | Using the complier’s highest-level warnings to check and modify the code until all warnings are eliminated. Using static and dynamic tools can help with eliminating the warnings and any security issues |
| 1. Architect and Design for Security Policies | Designing a code or application that forces the implementation of security policies. For example, designing the code to create different access levels based on the users’ job requirements. |
| 1. Keep It Simple | Creating a program or application using simple code to avoid security and quality issues. Complicated code leads to complex issues and operational errors. |
| 1. Default Deny | By default, the code or the application denies unauthorized access. Standard user has no access permissions without the administrator's approval. |
| 1. Adhere to the Principle of Least Privilege | Users only have the necessary permissions to do their jobs or finish tasks. Any elevated permissions are for a limited time until the task is done. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing and cleaning any data or inputs sent to sub-systems. The data could be commands or SQL queries. The sub-systems are not part of the validation process on the primary system, so it is essential to sanitize any data going to them. |
| 1. Practice Defense in Depth | Adding multiple layers of security to eliminate any security issues in the code. For example, validating input data and using the least privilege security principle to secure the code. |
| 1. Use Effective Quality Assurance Techniques | Using multiple quality insurance techniques like code testing, code auditing and penetration testing can early detect any bugs, error or security issues with code. |
| 1. Adopt a Secure Coding Standard | Using one or more of the previous security principles plus developing or adding other security methods will ensure the code security and quality are high. |

### 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] | Don’t change the reference type to const or volatile |

| **Noncompliant Code** |
| --- |
| p is the reference to const-qualified char which leads to syntax error. |
| #include <iostream>    void f(char c) {    const char &p = c;    p = 'p'; // Error: read-only variable is not assignable    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Removed the const qualifier from p. |
| #include <iostream>    void f(char c) {    char &p = c;    p = 'p';    std::cout << c << std::endl;  } |

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

| **Principles(s):** Validate input data and head complier warnings principles can be applied to this standard. The standard prevents the value change of refence type. Validating the input data and pay attention to the warnings will prevent this error which can lead to unidentified behavior and bug the entire code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 | NA |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-DCL52-a | Never qualify a reference type with ‘const’ or ‘volatile’ |
| Polyspace Bug Finder | R2023a | CERT C++: DCL52-CPP | Checks for: const-qualified reference types and modification of const-qualified reference types |
| Clang | 3.9 | NA | Checks for violations of the rule and produces an error |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Make sure division does not lead to divide by zero errors |

| **Noncompliant Code** |
| --- |
| The code will prevent overflow but will not prevent divide by zero error from dividing s\_a and s\_b |
| #include <limits.h>    void func(signed long s\_a, signed long s\_b) {    signed long result;    if ((s\_a == LONG\_MIN) && (s\_b == -1)) {      /\* Handle error \*/    } else {      result = s\_a / s\_b;    }    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The code will check for divide by zero error and overflow |
| #include <limits.h>    void func(signed long s\_a, signed long s\_b) {    signed long result;    if ((s\_b == 0) || ((s\_a == LONG\_MIN) && (s\_b == -1))) {      /\* Handle error \*/    } else {      result = s\_a / s\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** using scientize data sent to other systems and practice DiD principles will ensure that the division and reminder do not cause overflow. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | Int-division-by-zero  Int-modulo-by-zero | Fully checked |
| CodeSonar | 7.3p0 | LANG.ARITH.DIVZERO  LANG.ARITH.FDIVZERO | Division and float division by zero |
| Coverity | 2017.07 | DIVIDE-BY-ZEOR | Fully implemented |
| LDRA tool suite | 9.7.1 | 43 D, 127D, 248 S, 629 S, 80 X | Partially implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Creating a std::string from a null pointer is prohibited |

| **Noncompliant Code** |
| --- |
| In the example std::string is the result of calling std::getenv(). The call returns null and cause erros |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In the example the result of calling getenv() checked before constructing std::string |
| #include <cstdlib>  #include <string>    void f() {    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):** using effective quality assurance techniques can avoid the error in this standard by checking the pointer before assigning it to a string |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Assert\_failure | NA |
| CodeSonar | 7.3p0 | LANG.MEM.NPD | Null pointer dereference |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-STR51-A | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2023a | CERT C++: STR51-CPP | Check for string operations on null pointer |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Range Check element access |

| **Noncompliant Code** |
| --- |
| In the example, we are trying to replace the first letter in a string with a capital letter. If the string is empty, the code will have an error |
| #include <string>  #include <locale>    void capitalize(std::string &s) {    std::locale loc;    s.front() = std::use\_facet<std::ctype<char>>(loc).toupper(s.front());  } |

| **Compliant Code** |
| --- |
| In this example, the program checks if the string empty before calling std::string::front() |
| #include <string>  #include <locale>    void capitalize(std::string &s) {    if (s.empty()) {      return;    }      std::locale loc;    s.front() = std::use\_facet<std::ctype<char>>(loc).toupper(s.front());  } |

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

| **Principles(s):** adopting a secure code standard like validating data and DiD will check the code for this standard to ensure the called string is not empty or out of range. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Assert\_failure | NA |
| CodeSonar | 7.3p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer overrun  Buffer underrun  Tainted buffer access  Type overrun  Type unerrun |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-STR53-a | Ensures the container withing the valid range |
| Polyspace Bug Finder | R2023a | CERT C++: STR53-CPP | Checks for array access out of bounds or with tainted index and pointer dereference with tainted offset |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Detect and handle memory allocation errors to prevent program termination and denial of services attacks |

| **Noncompliant Code** |
| --- |
| In the example, the same expression was used to do two memory allocations. The memory allocations are arguments to a function call. The thrown exception due to one of the calls to ‘new’ could lead to a memory leak. |
| struct A { /\* ... \*/ };  struct B { /\* ... \*/ };    void g(A \*, B \*);  void f() {    g(new A, new B);  } |

| **Compliant Code** |
| --- |
| In the example, passed the objects by reference and deleted the memory allocation. |
| struct A { /\* ... \*/ };  struct B { /\* ... \*/ };    void g(A &a, B &b);  void f() {    A a;    B b;    g(a, b);  } |

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

| **Principles(s):** using quality assurance techniques like code testing will help eliminating the error from this standard and preventing denial of services attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds how function call return values gets handled |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Polyspace bug finder | R2023a | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Checks the return value of new |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Do not abruptly close the program to avoid keeping the resources busy. This could lead to denial of service attack |

| **Noncompliant Code** |
| --- |
| In the example, the code calls f() as exit handler using std::at\_exit. The call might go to std::terminate() which may throw an exception due to throwing\_func() |
| #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** |
| --- |
| In the example, any exception by throwing\_func() gets handled by f(). |
| #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):** paying attention to heed complier warnings and applying architect and design for security policies principle will help with closing the program in the right way to free the resources and prevent denial of services attacks |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Stdlib-use | Partiall checked |
| CodeSonar | 7.3p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Polyspace bug finder | R2023a | CERT C++: ERR50-CPP | Checks for implicit call to terminate() function |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions to ensure releasing all resources and avoiding denial of service attack |

| **Noncompliant Code** |
| --- |
| In the example, f() and main() can’t catch the thrown exceptions form throwing\_func() because there is no handler for the thrown exception from calling std::terminate() |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| In the example, all exceptions get handled from beginning before going to the main() function |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    try {      f();    } catch (...) {      // Handle error    }  } |

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

| **Principles(s):** using quality assurance techniques like error handles will ensure the continuity of the program in this standard and will help with finding any errors |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| CodeSonar | 7.3p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| RuleChecker | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | [STD-008-CPP] | Do not cast to an out-of-range enumeration value. It could lead to overflow. |

| **Noncompliant Code** |
| --- |
| The code in the example checks if the give value withing the range or not. the code checks the value assigning it to enumeration type which causes unspecified behavior |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| The code in the example will check the value withing the acceptable range before assigning it to enumeration type. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** sanitizing the data will eliminate the errors in this standard and prevent buffer overflow |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Cast-integer-to-enum | Partially checked |
| CodeSonar | 7.3p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion alters value  Cast alters value |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-INT50-a | An expression with enum type shall only have values corresponding to the enumerators |
| RuleChecker | 22.10 | Cast-integer-to-enum | Partially checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output (FIO) | [STD-009-CPP] | Close files when they are no longer needed to protect the system resources. |

| **Noncompliant Code** |
| --- |
| the code in the example constructed std::fstream called for opening the file but it did not call for closing the file properly |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| The code in the example used std::fstream::close to close the file properly |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    file.close();    if (file.fail()) {      // Handle error    }    std::terminate();  } |

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

| **Principles(s):** applying architect and design for security polices principle can help with the errors in this standard. The code design needs to have a method to ensure the closing of unused files to free the system resources. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-F1O51-a | Ensure resources are freed |
| Parasoft Insure++ | NA | NA | Runtime detection |
| Polyspace bug finder | R2023.a | CERT C++:F1O51-CPP | Checks for resource leak |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-010-CPP] | Do not delete an array through a pointer of incorrect type. It could lead to memory leak and abnormal program execution |

| **Noncompliant Code** |
| --- |
| In the example, the pointer of Derived object array stored in Base although Base is declared virtual causing undefined behavior |
| struct Base {    virtual ~Base() = default;  };    struct Derived final : Base {};    void f() {     Base \*b = new Derived[10];     // ...     delete [] b;  } |

| **Compliant Code** |
| --- |
| In the example, a static type of b was Derived to eliminate the undefined behavior when deleting the pointer |
| struct Base {    virtual ~Base() = default;  };    struct Derived final : Base {};    void f() {     Derived \*b = new Derived[10];     // ...     delete [] b;  } |

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

| **Principles(s):** applying keep it simple principle will help with fixing the issue in this standard. A simple code will ensure that the right pointer type is used to delete ana array |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -analyzer-checker=cplusplus | Cheked with clang -ccl or scan-build |
| CodeSonar | 7.3p0 | ALLOC.TM | Type mismatch |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-EXP51-a | Do not treat arrays polymorphically |
| Polyspace bug finder | R2023a | CERT C++:EXP51-CPP | Checks for delete operator used to destroy downcast object of different type |

### 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.

DevSecOps constructs an amazing base to apply security policies using automation tools. However, I think that using secure coding standard can add security layers to the pre-production process. for example, sanitizing and validating input data form or to other systems can help with eliminating SQL injections threats. Next, keep the code as simple as possible and using default deny helps avoiding complicated issues and unnecessary access. Also, using quality assurance techniques like automated code testing will help finding bugs and errors and avoid operational issues. On the production stage, we can IDS/IPS to monitor and take actions against any suspicious behaviors or known vulnerabilities. Finally, using defense in depth like multi-factor authentication will eliminate unauthorized access to the system.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-002-CPP | Low | Likely | Medium | P6 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Probable | Medium | P4 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Low | Unlikely | Medium | P2 | L3 |
| [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 | It helps in protecting data that saved on hard drive or a server. It is used to prevent attackers who gain access to a data on hard drive or a server from stealing or copying sensitive information |
| Encryption at flight | It is used to encrypt a transmitted data. For example, sending sensitive information through an email server. Only the receiver will have the key to decrypt the data. The encryption will protect the data if attacker in the middle-gained access to the email |
| Encryption in use | It is used to protect the data all the time. The encryption will protect the data in cases like unheroized access to the system. it is also used to protect data in motion and data in reset. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | A method to authenticate the user before allowing access to the system. it includes using a password or something you have like a USB key or something you are like fingerprints biometrics or a combination of all of them |
| Authorization | Authorizing the user access to specific folders or part of the system based on their permissions and job descriptions. For example, HR manager will need full access to the employees’ folders. On the other side, a coordinator will not need access to those folders. |
| Accounting | It is a way to monitor the user activity and blocking any suspicious behaviors. Also, it helps with finding the root cause for any security issues |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 04/03/2023 | Final Template | Mina Abadeer | [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 |

Reference

<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>