**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 this core security principle we make sure that any data that is being inputted is checked to ensure that it follows the proper format. |
| 1. Heed Compiler Warnings | Compiler warnings are used for letting the user know that something went wrong with the code. Depending on how the warning is programmed it can help the user know where the error is located. |
| 1. Architect and Design for Security Policies | This core security principle lays down the architecture of security for the system it helps with stating roles clearly for the system. |
| 1. Keep It Simple | This core security principle is about keeping the programs code simple. Over complicating the code means complicating security. |
| 1. Default Deny | Default deny ensure that anything that you have not allowed to pass through the system will be denied by default. |
| 1. Adhere to the Principle of Least Privilege | This security principle gives you access to only the areas which you need to work with and any other area which you wont work with wont allow access. |
| 1. Sanitize Data Sent to Other Systems | By sanitizing data you are removing any unwanted characters that may cause harm the other system which is receiving the data. |
| 1. Practice Defense in Depth | Defense in depth layers multiple layers of security so that if one fails there is a backup creating a security by depth. |
| 1. Use Effective Quality Assurance Techniques | Making sure that the code that is being written keeps a standard of quality. |
| 1. Adopt a Secure Coding Standard | A secure coding standard allows the users to keep a uniform and expected way of programming. |

### 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** | DCL52-CPP | Never qualify a reference type with const or volatile. |

| **Noncompliant Code** |
| --- |
| P is declared to be const but the variable is later assigned a value. |
| void f(char c) {  char &const p = c;  p = 'p';  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| In this code block the const qualifier is removed. |
| 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):** For this Coding Standard Principles 9 and 10 apply. Principle 9 deals with quality in the code which this standard helps with keeping good quality coder and for Principle 10 is for keeping a coding standard which this Standard applies. |
| --- |

**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 | N/A |
| Helix QAC | 2023.1 | C++0014 | N/A |
| Klocwork | 2023.1 | CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE | N/A |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL52-a | Never qualify a reference type with 'const' or 'volatile' |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT50-CPP | Do not cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| Check bounds after casting the enumeration type. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| Checks that the value is able to be represented by the type by checking bounds. |
| 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):** For this Principle 1, 7, 9 and 10 apply. The principle 1 and 7 apply because if the numeration is out of range then the data inputted or sent to other system will have unexpected results. As for 9 and 10 this allows to keep the quality and standardization. |
| --- |

**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 |
| Partially checked | 7.2.0 | CertC++-INT50 | N/A |
| CodeSonar | 7.4p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2023.1 | C++3013 | N/A |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR50-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

| **Noncompliant Code** |
| --- |
| Input is unbounded which can lead to buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Use of std::string which guards against buffer overflow. |
| #include <iostream>  #include <string>    void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** For this Principle 1, 7, 9 and 10 apply. The principle 1 and 7 apply because when dealing with strings if they are sent to another system without proper formatting it may cause unexpected results and the same goes if it isn’t validated. As for 9 and 10 this allows to keep the quality and standardization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| RuleChecker | 22.10 | stream-input-char-array | Partially checked |
| Helix QAC | 2023.1 | C++5216  DF2835, DF2836, DF2839, | N/A |
| Klocwork | 2023.1 | [Insert text.] NNTS.MIGHT  NNTS.TAINTED  NNTS.MUST  SV.UNBOUND\_STRING\_INPUT.CIN | N/A |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STR53-CPP | Range check element access. |

| **Noncompliant Code** |
| --- |
| The value returned by get\_index() may be greater than what is stored in the string. |
| #include <string>    extern std::size\_t get\_index();    void f() {  std::string s("01234567");  s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| Throws a out of range exception if the position is greater than or equal to the size. |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();    void f() {  std::string s("01234567");  try {  s.at(get\_index()) = '1';  } catch (std::out\_of\_range &) {  // Handle error  }  } |

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

| **Principles(s):** For this Principle 1, 7, 9 and 10 apply. The principle 1 and 7 apply because if the range isnt checked it may cause unexpected results leading to a possibility of injection. As for 9 and 10 this allows to keep the quality and standardization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | assert\_failure | [Insert text.] |
| CodeSonar | 7.4p0 | 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 underrun |
| Helix QAC | 2023.1 | C++3162, C++3163, C++3164, C++3165 | N/A |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM50-CPP | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| After s is deallocated it is dereferenced which can be exploited if the access results in a write-after-free. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):** For this Principle 9 and 10 apply. Principles 9 and 10 allows us to keep the quality and standardization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | dangling\_pointer\_use | [Insert text.] |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM50 | [Insert text.] |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 7.4p0 | ALLOC.UAF | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | CTR54-CPP | Do not subtract iterators that do not refer to the same container |

| **Noncompliant Code** |
| --- |
| This code leads to undefined behavior because of the use of std::less() instead of <. |
| #include <functional>  #include <iostream>    template <typename Ty>  bool in\_range(const Ty \*test, const Ty \*r, size\_t n) {  std::less<const Ty \*> less;  return !less(test, r) && less(test, r + n);  }    void f() {  double foo[10];  double \*x = &foo[0];  double bar;  std::cout << std::boolalpha << in\_range(&bar, x, 10);  } |

| **Compliant Code** |
| --- |
| This implementation is portable and it compares test against each possible address in range. |
| #include <iostream>    template <typename Ty>  bool in\_range(const Ty \*test, const Ty \*r, size\_t n) {  auto \*cur = reinterpret\_cast<const unsigned char \*>(r);  auto \*end = reinterpret\_cast<const unsigned char \*>(r + n);  auto \*testPtr = reinterpret\_cast<const unsigned char \*>(test);    for (; cur != end; ++cur) {  if (cur == testPtr) {  return true;  }  }  return false;  }    void f() {  double foo[10];  double \*x = &foo[0];  double bar;  std::cout << std::boolalpha << in\_range(&bar, x, 10);  } |

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

| **Principles(s):** For this Principle 9 and 10 apply. Principles 9 and 10 allows us to keep the quality and standardization.s |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | invalid\_pointer\_subtraction  invalid\_pointer\_comparison | N/A |
| CodeSonar | 7.4p0 | LANG.STRUCT.CUP  LANG.STRUCT.SUP | Comparison of Unrelated Pointers  Subtraction of Unrelated Pointers |
| Helix QAC | 2023.1 | DF2668, DF2761, DF2762, DF2763, DF2766, DF2767, DF2768 | N/A |
| LDRA tool suite | 9.7.1 | 70 S, 87 S, 437 S, 438 S | Enhanced Enforcement |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR56-CPP | Guarantee exception safety. |

| **Noncompliant Code** |
| --- |
| This implementation uses a flawed copy assignment. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }      IntArray(const IntArray& that); // nontrivial copy constructor  IntArray& operator=(const IntArray &rhs) {  if (this != &rhs) {  delete[] array;  array = nullptr;  nElems = rhs.nElems;  if (nElems) {  array = new int[nElems];  std::memcpy(array, rhs.array, nElems \* sizeof(\*array));  }  }  return \*this;  }    // ...  }; |

| **Compliant Code** |
| --- |
| This implementation has a strong exception safety in the copy assignment oprator. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }    IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {  int \*tmp = nullptr;  if (rhs.nElems) {  tmp = new int[rhs.nElems];  std::memcpy(tmp, rhs.array, rhs.nElems \* sizeof(\*array));  }  delete[] array;  array = tmp;  nElems = rhs.nElems;  return \*this;  }    // ...  }; |

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

| **Principles(s):** For this Principle 9 and 10 apply. Principles 9 and 10 allows us to keep the quality and standardization. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2023.1 | C++4075, C++4076 | N/A |
| LDRA tool suite | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR56-a  CERT\_CPP-ERR56-b | Always catch exceptions  Do not leave 'catch' blocks empty |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Expressions** | EXP53-CPP | Do not read uninitialized memory. |

| **Noncompliant Code** |
| --- |
| The uninitialized variable is done inside expression which can cause undefined behaviour. |
| #include <iostream>    void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| In this implementation the variable is initialized. |
| #include <iostream>    void f() {  int i;  std::cout << i;  } |

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

| **Principles(s):** For this Principle 9 and 10 apply. Principles 9 and 10 allows us to keep the quality and standardization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | uninitialized-read | Partially checked |
| Clang | 3.9 | -Wuninitialized  clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| CodeSonar | 7.4p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local  Uninitialized variable |
| Helix QAC | 2023.1 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 | N/A |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Integers** | INT50-CPP | Do not cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| Checks whether the value is within range but it does it after the enumeration type is casted. |
| 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 check is done before the conversion is done. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

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

| **Principles(s):** For this Principle 1, 7, 9 and 10 apply. The principle 1 and 7 apply because if the numeration is out of range then the data inputted or sent to other system will have unexpected results. As for 9 and 10 this allows to keep the quality and standardization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | cast-integer-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0s | CertC++-INT50 | N/A |
| CodeSonar | 7.4p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2023.1 | C++3013 | N/A |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Declarations** | DCL58-CPP | Do not modify the standard namespaces. |

| **Noncompliant Code** |
| --- |
| Declaration x is added to the namespace std. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| The declaration is placed into namespace without reserved name. |
| namespace nonstd {  int x;  } |

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

| **Principles(s):** For this Principle 9 and 10 apply. Principles 9 and 10 allows us to keep the quality and standardization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL58 | N/A |
| CodeSonar | 7.4p0 | LANG.STRUCT.DECL.SNM | Modification of Standard Namespaces |
| Helix QAC | 2023.1 | C++3180, C++3181, C++3182 | N/A |
| Klocwork | 2023.1 | CERT.DCL.STD\_NS\_MODIFIED | N/A |

### 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.

The process of automation should take place in the last possible section of the production stage. Before the product is sent to the customer having it thoroughly tested with automation programs will help with having a solid product for the customer.

### 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 |
| --- | --- | --- | --- | --- | --- |
| DCL52-CPP | Low | Unlikely | Low | P3 | L3 |
| INT50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STR50-CPP | High | Likely | Medium | P18 | L1 |
| STR53-CPP | High | Unlikely | Medium | P6 | L2 |
| MEM50-CPP | High | Likely | Medium | P18 | L1 |
| CTR54-CPP | Medium | Probable | Medium | P8 | L2 |
| ERR56-CPP | High | Likely | High | P9 | L2 |
| EXP53-CPP | High | Probable | Medium | P12 | L1 |
| INT50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| DCL58-CPP | High | Unlikely | Medium | P6 | L2 |

### 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 not being used or sent to another location then it is considered to be at rest. When applying encryption at rest we are looking to safe guard the files that are located in the database to prevent them from being accessed by users who shouldn’t. |
| Encryption at flight | When data is being sent to another location it is at flight. When data is at flight we are looking to encrypt both ends so that only the one who receives it is able to access the information that is being and received and the person who sends it protects against those looking to access the information as a middle man. |
| Encryption in use | When data is at use then we are actively using the data that was received. When applying encryption to data in use we are looking to prevent users from accessing the data we are looking at. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | For this step we are ensuring that the person who is trying to access the system is that person. This can be done by using user logins which hold a username and password which can be used for log in. |
| Authorization | After the authentication is done you enter the authorization where you have access only to the features you need access to if you have the proper authorization you can make changes to the database, add new users and modify user access. |
| Accounting | With this step those who make changes to the system hold the accountability for those actions by saving the action alongside the username of the user |

**\***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 | 07/16/2023 | Milestone 1 | Jehu Domenech |  |
| 1.2 | 08/06/2023 | Project 1 | Jehu Domenech |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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