**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** |  |
| --- | --- |
| 1. ValidateInput Data | Only will allow suitable data. Ensures that disallowable input does not enter the system and will help prevent any harmful data from being inputted. This is process helps to ensure the data that is being entered is clean and correct. |
| 1. Heed Compiler Warnings | The highest level of warnings should be used with any compiler. If a warning exists it is important to eliminate this warning as if they are not dealt with properly, they can lead to security issues later. |
| 1. Architect and Design for Security Policies | These are policies that ultimately align with the goals/ objectives of the system. This is the software architect as well as the design. Security architects help with predicting any potential threats and what designs will help prevent them. Software architect should be built to implement/ enforce security policies. |
| 1. Keep It Simple | Keeping your code simple/ small is very important when developing code. Your code should be easy to read, understand, and update. This is keeping variable names simple but also purposeful. Developing too complex code can leave the door open for potential errors and security issues. |
| 1. Default Deny | This focuses on only allowing authorized access and denying access by default. Protection schemes will ultimately deem the permission as authorized. This means that unless specifically stated that access is allowed it will be denied. |
| 1. Adhere to the Principle of Least Privilege | The system should perform with the least number of privileges that are required. Any excess privileges that are used should only be used for the minima amount of time necessary. In doing so we can limit the chance of vulnerabilities being accessed as well as the amount of damage done by an attacker from unauthorized use. |
| 1. Sanitize Data Sent to Other Systems | This is the process of sanitizing any and all data that is being passed. Data can come from multiple different complex subsystems which gives an attacker multiple ways to hack the system through injection attacks. BY sanitizing data before using the subsystem we can help prevent these attacks from happening. |
| 1. Practice Defense in Depth | Defense in Depth (DiD) is a security strategy that utilizes multiple security measures to try and keep sensitive information protected. This is the principle of having multiple defensive strategies in place. IF one layer of defense is corrupted there is another layer of defense to attempt to prevent security issues. |
| 1. Use Effective Quality Assurance Techniques | Use effective testing techniques to help identify and eliminate possible vulnerabilities. Utilizing multiple testing processes will create an effective quality assurance technique. Having proper testing can help to lead to having a secure system. |
| 1. Adopt a Secure Coding Standard | Develop and utilize standard coding best practices. These change based on the language and platform being used. In C++ these standards are the principles listed as well as resources such as SEI CERT C++ coding standards. |

### 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.

[SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682)

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| If we cast to an out of range enumeration the value may lead to an unspecified value/ undefined actions. |
| Enum {Small,Medium,Large};    void function(int intVal) {  Enum x=static\_cast<Enum>(intValue);    if (Value=Small || Value=Medium || Value=Large) {  }  } |

| **Compliant Code** |
| --- |
| To avoid leading to an unspecified value we check the value before casting. This will ensure that intVal is in range which will in turn prevent any undefined actions. |
| Enum {Small,Medium,Large};  void function(int intVal) {  if (intValue>=Small && intValue<=Large) {  Enum x=static\_cast<Enum>(intValue);  }  } |

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

| **Principles(s):** Validate Input Data. Ensure we are validating the data that is being used which can help to prevent potential buffer overflows. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.1 | Cast-integer-to-enum | Partly Checked |
| Rule Checker | 2201 | Cast-integer-to-enum | Partly Checked |
| Helix QAC | 2024.2 | C++30113 |  |
| PVS Studio | 7.33 | V1016 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use valid references, pointers, and iterators to reference elements of a container |

| **Noncompliant Code** |
| --- |
| An error with positoin will occur after the first iteration as it will be nullified after the first call is made to insert(). Any other loop iterations will have undefined actions. |
| void function(const int \*s, counter) {  <int> x;  position = x.reserve();  for (j = 0; j < count; ++j, ++position) {  x.resize(position, s[i] + 21.0);  }  } |

| **Compliant Code** |
| --- |
| In this code example position has a valid iterator for each insertion which will prevent undefined actions. |
| void function(const int \*s, counter) {  <int> x;  position = x.reserve();  for (j = 0; j < count; ++j, ++position) {  position = x.resize(position, s[j] + 21.0);  }  } |

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

| **Principles(s):** Secure Coding Standards. It would be considered bad practice to not use valid references and can cause unknown actions later throughout the execution of the code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSnar | 8.1p0 | ALLOC.UAF | Pointer before each insertion |
| Polyspace Bug Finder | R2024a | CERT C++: STR52-CPP | Checks that values are valid before insertions |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-STR52a | Checked for invalid strings iterations |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Range check element access |

| **Noncompliant Code** |
| --- |
| May result in undefined actions if the value returned from get\_index() is greater than the what is within the string. |
| get\_index();    void function() {  string y("hello");  y[get\_index()] = '3';  } |

| **Compliant Code** |
| --- |
| Will check that the value returned from get\_index() is within the number of elements from the string. |
| get\_index();    void function() {  string y("hello");  size\_x j = get\_index();  if (j < y.length()) {  s[j] = '3';  } else {  }  } |

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

| **Principles(s):** Validate Input Data and Secure Coding Standards. We want to check that the value returned is within the elements from the string. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.1 | Assert\_failue |  |
| CodeSonar | 8.1p0 | LANG.MEM. | Checks for different buffers (over/ under) |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP=STR53-a | Guarantees the index is within the given range |
| Polyspace Bug Finder | R2024a | CERT C++: STR53-CPP | Checks for accesses with an array as well as pointer refrences |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not store an already-owned pointer value in an unrelated smart pointer |

| **Noncompliant Code** |
| --- |
| In this example s and s1 are unrelated pointers but are storing the same pointer “t”. This can lead to issues when calling functions related to the pointer. |
| void function() {  double \*t = double;  shared\_ptr<double> s(t);  shared\_ptr<double> s1(t);  } |

| **Compliant Code** |
| --- |
| In this example s and s1 are related to each other as s1 is only created from copying s. This will eliminate the issues as it exists in two related smart pointers. |
| void function() {  shared\_ptr<double> s = make\_shared<double>();  shared\_ptr<double> s1(s1);  } |

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

| **Principles(s):** Architecture and Design for Security Polices and Heed Compiler Warnings. The issue will cause an error/ warn about storing the same pointer which should not be ignored. This standard is being used to prevent an SQL injection attack from happening. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.33 | V1006 |  |
| Ployspace Bug Finder | R2024a | CERT C++: MEM56-CPP | Will check the pointer has not already been used |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-MEM56-a | Don’t store a pointer value that is already owned in and unrelated pointer |
| Axivion Bauhaus Suite | 7.2 | CERC++-MEM56 |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| S has been dereferenced after already being deallocated. This can result in vulnerabilities being exploited as S will ultimately be free to write to. |
| struct T {  void v();  };    void function() {  T \*t = new T;  t->v();  } |

| **Compliant Code** |
| --- |
| To eliminate this potential vulnerability, we need to wait until the allocated memory is not deallocated until it is no longer needed. |
| struct T {  void v();  };    void function() {  T \*t = new T;  s->v();  remove t;  } |

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

| **Principles(s):** Secure Coding Standards and Heed Compiler Warnings. If a warning is shown due to accessing freed memory it should not be ignored as it could lead to the application crashing. If not taken care of there is a chance of accessing memory not available. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Clang-anaylyzer-cplusplus.NewDelteclang-anaylyzer-alpha.security.ArrayBound | Utilized clang to catch some violations |
| CodeSonar | 8.1p0 | ALLOC.UAF | Will use memory after it has been freed |
| Parasoft C/C++ tets | 2023.1 | CERT\_CPP-MEM50-a | Does not use memory that has been freed |
| Coverity | V7.5 | USE\_AFTER\_FREE | Once instances have been freed or memory has been deallocated |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CGL] | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| In this example we use the assert() which is typically used to handle errors that will test for a condition that is not true. Therefore the use of assert() here is not correct. |
| struct time {  unsigned string one;  unsigned double two;  }    int function(void) {  assert(sizeof(struct time) == sizeof(unsigned string) / sizeof(unsigned double) + sizeof(unsigned double));  } |

| **Compliant Code** |
| --- |
| When using static\_assertion a preprocessor conditional statement may be used. It will allow for incorrect assumptions that will be recognized once compiled. |
| struct time {  unsigned string one;  unsigned double two;  }    static\_assert(sizeof(struct time) == sizeof(unsigned string) / sizeof(unsigned double) |

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

| **Principles(s):** Secure Coding Standards and Effective Quality Assurance Techniques. We are utilizing tests to test the value of a const expression. This can help eliminate many defects that may arise. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 8.1p0 | Clang-anaylyzer-cplusplus | Can utilize different checker to notify the use of assert() |
| Axivion Bauhuas Suite | 3.9 | Different checkers can be created | [Insert text.] |
| Compass/ROSE | 4.1.2 |  | Can detect various violations of the rule |
| ECLAIR | 1.2 | CC2.DCL03 | Completely Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions thrown before main() begins executing |

| **Noncompliant Code** |
| --- |
| We may potentially have an exception thrown that will not be caught as globalT is only created when the program is starting up. |
| struct T {  T();  };    static T globalT; |

| **Compliant Code** |
| --- |
| We can change this and make globalT a local variable that will allow exceptions that are being thrown to be caught as when the constructor T will be run it is the first time globalT() will be called. |
| struct T {  T();  };    T globalT() {  try {  static T t;  return t;  } catch {  }  } |

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

| **Principles(s):** Heed Compiler Warnings, Effective Quality Assurance and Techniques and Secure Coding Standards. If a warning is thrown for an exception, it should be dealt with appropriately. All the exceptions should be caught when thrown. Utilizing testing can help find the issues and understand how to fix them. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 7.2 | Cert-err58-cpp | Utilized clang to check all exceptions are handled |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-ERR58-a | Exceptions will only be thrown after the program is started and before the program is ended |
| Polyspace Bug Finder | R2024a | CERT C++: ERR58-CPP | Looks for any exceptions during the program being started |
| RuleChecker | 22.1 | Potentially-throwing-static-initialization | Will partly check the exceptions being thrown |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | [STD-008-CPP] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| There are no exceptions thrown in this example as there are no matching handler for the exceptions being thrown. |
| void throw\_function();    void function() {  throw\_function();  }    // main function will call function() |

| **Compliant Code** |
| --- |
| Will handle all of the exceptions that will be thrown. Using the main function and try/ catch will help to manage external sources and catch these exceptions. |
| void throw\_function();    void function() {  throw\_function();  }  // main function to use try and catch blocks |

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

| **Principles(s):** Develop Coding Standards and Defense in Depth. Programs should handle any and all exceptions to ensure nothing has been compromised during the execution. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 220.1 | Main-function-catch-all-early-catch-all | May be able to partly catch the exceptions |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-ERR51-a | Catch the exceptions every time one is thrown. |
| Polyspace Bug Finder | R2024a | CERT C++: ERR51-CPP | Will check for any unhandled exceptions |
| LDRA tool suite | 9.7.1 | 527 S | Will partially check for all exceptions |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-009-CPP] | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| The local variable is not initialized and will be assessed as part of an expression when printing the value “j”. This may result in undefined behavior. |
| void function() {  double j;  cout << j;  } |

| **Compliant Code** |
| --- |
| To combat this we initialize the variable before calling the print function to print its value. |
| void funtion() {  double j = 2;  cout << j;  } |

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

| **Principles(s):** Secure Coding Standards. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wuninitialized-clang-analyzer-core.UndefinedBinaryOperatorResult | Will not catch everything. |
| CodeSonar | 8.1p0 | LANG.STRUCT.RPL | Will return a pointer to a local variable |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-EXP53-a | Will avoid any use of memory before being intialized |
| Polyspace Bug Finder | R2024a | CERT C++: EXP53-CPP | Will check for both uninitialized pointers and variables. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Strings | [STD-010-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| The input is not bounded which can lead to potential buffer overflows. |
| void function() {  string buffer[];  cin >> “the buffer is “ >> buffer;  } |

| **Compliant Code** |
| --- |
| Instead of using an array it is better to use string to prevent any potential buffer overflows. |
| void function() {  string choice, choice1;  cin >> choice >> choice1;  } |

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.MEM.BO/TO  MISC.MEM.NTERM | Checks for buffer overflows as well as doesn’t allow null terminators |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-STR50-b | Avoids buffer overflows as it will not read null strings or contaminate data. |
| Polyspace Bug Finder | R2024a | CERT C++: STR50-CPP | Checks for a variety of issues to ensure storage for strings and null terminator |
| Visual Studio Checker | 2022 | Standard | Is standard checker available with visual studio download that will throw errors |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

Automation will be utilized for the enforcement and the compliance to the standards that are defined within this policy. An IDE security plugin tool that can be used is Snyk. Snyk is compatible with multiple different IDE’s and can offer real-time vulnerability assessments during the building phase. We can also utilize Static Application Security Testing (SAST). SAST is used during the verification phase and will scan the code to help to identify vulnerabilities. It will also provide instant results to developers. Within the monitor and detect phase and automation tool that can be utilized it RASP (Runtime Application Self-Protection). RASP security can keep the application protected as it monitors the application as it runs. A specific RASP software includes Aikido. RASP tools can help protect the application and give developers insight into attacks. RASP can also be utilized during the respond phase as it will help the developers understand the attacks and prevent others from happening. Utilizing automation can greatly improve the applications security. Technology is a much more trustworthy option than just humans. Humans can make a mistake and not recognize it but with the use of automation tools these mistakes can be found and mitigated quickly. Automation can be used throughout various phases of the DevOps process and infrastructure.

### 

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

### 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 | Encryption at rest consists of data that is on something like a disk or hard drive. This data is stored on different devices and not within the system itself. The policy applies as sensitive data should be stored separate from the system to ensure its confidentiality. This policy is helpful at preventing any unauthorized access from taking place. The data is also encrypted and has to be decrypted to access any of the information. Some helpful things to do to keep this data protected can include securing the infrastructure, make sure users understand security risks, and encrypting the data. |
| Encryption in flight | Encryption in flight refers to data that is transferred throughout networks. This policy issued to protect the data as it is being sent between two different networks. The data is encrypted as it is being sent and is decrypted once it is received by the authorized user. Both the send and receiver utilize and encryption key to accomplish this. This policy applies as there is a lot of sensitive data that is sent this way including banking, and emails. To keep this data secure there are many things that can be done such as controlling the access and encrypting the data. Keeping the data restricted based on roles can be helpful in controlling access. We can also utilize TLS (Transport Layer Security)/ SSL (Secure Sockets Layer). |
| Encryption in use | Encryption in use refers data that is actively being used. This policy is designed to protect the data while it is being accessed/ used. This policy applies as its purpose is to protect this data from any kind of unauthorized access. When data is being utilized it is the most vulnerable to security risks. To help keep this data secure some helpful things include data masking, controlling access, and monitoring. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication ensures only the appropriate people are accessing any given amount of information. Users typically will have credentials to login and prove the user is valid to access files. This login typically consists of usernames and passwords that will grant them access to the program. The reason this policy applies is we only want verified users to have access to any potential vulnerable data/ files. Authentication will verify the user’s identity and what the are allowed to access. It will help prevent and unauthorized access into programs. |
| Authorization | Authorization will verify the amount of access given to a user. This ensures users do not have access to all the files, data, and resources associated with a program. This policy applies because permissions should only be given to the appropriate role-based user. If we limit the accessibility with authorization, it helps keep the information of the program safer and less likely to be taken advantage of. |
| Accounting | Accounting is how we can track any activities/ changes happening within the system. We can see who is accessing what and when. Having the ability to do this we can see if any suspicious activity may be happening. This policy applies as it is important to monitor what is happening and being aware of odd behaviors. This can also be beneficial after any kind of attacks. If a user was accessing the information before it was leaked it can help element the breach quickly. |

**\***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 | 10/5/2024 | Milestone | Shaelyn Turner |  |
| 3.0 | 10/12/2024 | Project | Shaelyn Turner |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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