**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 | All input that is entered needs to be checked to ensure it is valid, but also needs to be sanitized in order to check for SQL injection threats. |
| 1. Heed Compiler Warnings | Compiler warnings can indicate a potential issue within the code that may cause running errors or unexpected results; compiler warnings can also indicate there are vulnerabilities within the code that have the potential for a security issue. |
| 1. Architect and Design for Security Policies | All policies should be reflected in the overall design of the application as well as the architecture. That means that security policies should be considered at the beginning of the development process. |
| 1. Keep It Simple | Use best practice simpler code instead of overcomplicated code and solutions. When you overcomplicate solutions or programs, there is more chance for errors within the code, meaning things can go awry, which means vulnerabilities can be added without knowledge, which means there are chances for threats and hacks. |
| 1. Default Deny | This means access and privileges to programs are automatically denied. They become granted only when required. |
| 1. Adhere to the Principle of Least Privilege | This means the lowest level of privilege needed to complete a task is what should be given. This helps ensure security throughout the build process and after. It keeps users and admins constrained to their needed area, lowering the chance of potential issues. |
| 1. Sanitize Data Sent to Other Systems | When sending data to another system, we should make sure all data is valid and is formatted to the receiving system. By doing this, it means there are less risks with the data. |
| 1. Practice Defense in Depth | DiD is when you use layers of security instead of just authentication or verification. Using multiple layers adds redundancies to the system so if one layer fails or is hacked there are multiple other layers that are still in place. |
| 1. Use Effective Quality Assurance Techniques | Using effective quality assurance techniques makes sure any issues can be found and fixed prior to them being found or exploited. |
| 1. Adopt a Secure Coding Standard | Secure coding standards help deliver properly secured code. This ensure consistency and that the use of best practices are followed. |

### 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] | Obey one-definition rule |

| **Noncompliant Code** |
| --- |
| Two different translation units define a class of the same name with differing definitions. |
| // a.cpp  Struct S {  Int a;  };  // c.cpp  Class S {  Public:  Int a;  }; |

| **Compliant Code** |
| --- |
| Use of a header file to introduce the object into both translation units. |
| // S.h  struct S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):** 3: Architect and Design for Security Policies  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | type-compatibility  definition-duplicate  undefined-extern  undefined-extern-pure-virtual  external-file-spreading  type-file-spreading | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Fully Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | A function that returns a value must return a value from all code paths else it will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| All code paths do not return a value |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| All code paths now return a value |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  d} |

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

| **Principles(s):** 2 – Heed Compiler Warnings  4 – Keep It Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2023.1 | DF2888 |  |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | Fully implemented |
| Polyspace Bug Finder | R2023a | CERT C++: MSC52-CPP | Checks for missing return statements (rule partially covered) |
| SonarQube C/C++ Plugin | 4.10 | S935 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| Std::getenv returns a null pointer on failure which can lead to undefined behavior. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| The results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #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):** 2: Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert\_failure |  |
| Helix QAC | 2023.1 | C++4770, C++4771, C++4772, C++4773, C++4774 |  |
| Klocwork | 2023.1 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Software vulnerability is ensured when using complex subsystems that contain special characters that can trigger commands or actions. This is why it’s important to sanitize all string data. |

| **Noncompliant Code** |
| --- |
| Application inputs email address to buffer then uses string as argument |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| Uses allow-listing approach to sanitize data to define acceptable characters via a list and removed unwanted characters. |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  "1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {  \*cp = '\_';  } |

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

| **Principles(s):** 1. Validation Input Data  4. Keep It Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 |  | Supported by stubbing/taint analysis |
| Coverity | 6.5 | TAINTED\_STRING | Fully implemented |
| LDRA tool suite | 9.7.1 | 108 D, 109 D | Partially implemented |
| Polyspace Bug Finder | R2023a | CERT C: Rec. STR02-C | Checks for: • Execution of externally controlled command • Command executed from externally controlled path • Library loaded from externally controlled path Rec. partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | No access to freed memory |

| **Noncompliant Code** |
| --- |
| S has been dereferenced after deallocation. Can be exploited if it results in a write-after-free code. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| The allocated memory is not deallocated until it’s no longer required |
| #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):** 2: Heed Compiler Warnings  5: Default Deny  6: Adhere to the Principle of Least Privilege  9: Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Clang-tidy does not catch all violations of this rule. |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Detects specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Parasoft Insure++ |  |  | Runtime detection |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| Assert() to property concerning memory-mapped structure that is needed for the code to function properly |
| #include <assert.h>    struct timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    **int** func(void) {  **assert**(sizeof(struct timer) == sizeof(unsigned **char**) + sizeof(unsigned **int**) + sizeof(unsigned **int**));  } |

| **Compliant Code** |
| --- |
| A preprocessory conditional statement is used |
| struct timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | - |
| Clang | 3.9 | misc-static-assert | Clang-tidy |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exceptions thrown by the application must be caught by the matching exception handler. |

| **Noncompliant Code** |
| --- |
| F() and main() catch exceptions thrown by throwing func(). |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  f();  } |

| **Compliant Code** |
| --- |
| Main entry point handles all exceptions, this ensures stack is unbound to 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):**  3. Architect and Design for Security Policies  9. Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Klocwork | 2023.1 | MISRA.CATCH.ALL |  |
| Polyspace Bug Finder | R2023a | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| RuleChecker | 22.10 | main-function-catch-all early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declarations and Initialization | [STD-008-CPP] | When using namespaces std or posix you cannot add declarations or definitions. You also cannot do so with namespaces contained within those. |

| **Noncompliant Code** |
| --- |
| Declaration of x was added to namespace std. This shows undefined behaviors |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| Placing the declaration of x into namespace prevents collisions with other identifiers. |
| Namespace nonstd {  Int x;  } |

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

| **Principles(s):** 3. Architect and Design for Security Policies  4. Keep It Simple  10. Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL58 |  |
| Helix QAC | 2023.1 | C++3180, C++3181, C++3182 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL58-a | Do not modify the standard namespaces 'std' and 'posix' |
| PVS-Studio | 7.25 | V1061 |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | [STD-009-CPP] | Std::basic\_filebuf<T>::open() must be matched to a call std::basic\_filebuf<T>::close() |

| **Noncompliant Code** |
| --- |
| In this code the function is called to open but does not call to close |
| #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** |
| --- |
| Function is called to close before terminate making sure all files are closed properly prior to ending program |
| #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):** 5. Default Deny  6. Adhere to the Principle of Least Privilege |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  |  |
| Coverity | 2017.07 | RESOURCE\_LEAK (partial) | Partially implemented |
| LDRA tool suite | 9.7.1 | 49 D | Partially implemented |
| PC-lint Plus | 1.4 | 429 | Partially supported |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-010-CPP] | Member initializers are required in canonical order. |

| **Noncompliant Code** |
| --- |
| C::C() initializes someVal first then dependsOnSomeVal. This results in unspecified value saved in dependsOnSomeVal. |
| class C {  int dependsOnSomeVal;  int someVal;  public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| Changes declaration order of class member variables that way dependency is ordered correctly in constructors initializer list. |
| class C {  int someVal;  int dependsOnSomeVal;  public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

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

| **Principles(s):** 4. Keep It Simple  10. Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wreorder |  |
| CodeSonar | 7.4p0 | LANG.STRUCT.INIT.OOMI | Out of Order Member Initializers |
| RuleChecker | 22.10 | initializer-list-order | Fully checked |
| SonarQube C/C++ Plugin | 4.10 | S3229 |  |

### 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.

I believe that DevSecOps will help ensure when creating programs that we are creating the most secure application with the help of automation. We have outlined the policies and standards set within the security policy document and they should be utilized in the DevSecOps process. The first area where automation can make a difference will be when the building, testing and verifying steps come into play. CI/CD pipelines will be used to automate not only the building but also the testing of the application. This will help with security by making sure that steps are followed consistently when deploying code which allows security of the pipeline to be checked thoroughly.

CI/CD will have automation tools within the pipeline that can run and report unit tests, integration tests, and even front end UI tests. It will also allow for end to end testing to be created for the application. The tools will be able to check for dependency vulnerabilities as well.

When building and testing are finally completed, the CI/CD pipeline, after transition and health check steps, can automate the secure configuration and deployment of the application. After deployment, tools can automatically run penetration testing to check for weak or vulnerable spots in the code, security, or system.

Tools will be used to gather logs that will be output from the application that will monitor for any events. When issues are detected, alerts will be triggered, and admins will be alerted.

When thinking about maintaining and stabilizing steps automated tools can also be utilized to check integrity of the system and continue monitoring for any disturbances.

### 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 | P3 | L3 |
| STD-002-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |

### 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 | Data is in an encrypted state while contained in storage. What this means is even if access to data is gained the data is still encrypted or unreadable without the key for decryption. This policy applies in case there is ever a breach the information that can potentially be stolen is still safe since it cannot be read without the key. |
| Encryption in flight | Data that is encrypted while moving from one system or place to another. This makes sure that even when data is moved between systems, programs, end users, or anything like that the data is still safe and secure since it is encrypted. That way if the data is stolen in transit it is still safe. |
| Encryption in use | Data that is in use is still encrypted. This ensures that data is kept safe at all times even when in use or changing in the system. This is important as it ensures data is safe and secure within it’s life cycle, since it’s already protected when at rest or store, when in transit or moving you need to make sure it’s also safe as it’s being entered, changed, or in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is when you confirm a user is who they say they are. Authentication ensures that when someone want access to information the system ensures it knows who is asking for access. This is typical when adding new users or when users are gaining access to the system. This will apply when someone is trying to login or access the system or something within the system that has authentication required. This can be done with a username/password or a combination of both. |
| Authorization | This is when you confirm a user is allowed access to what they are trying to gain access to. Each user will typically have a level of access created when assigning usernames or admins. Authorization will help determine who has access to what and at what level. This applies when users are attempting to access the application or parts of the application. First will be the authentication to ensure the user is who they say they are then the authorization will check to see if they have clearance to what they are requesting. |
| Accounting | This is the tracking and logging of requests or the whereabouts of a user. This policy applies because it helps you keep track of what is happening with the system, who is requesting what information, and when and if they are allowed to request that information. This also tracks changes made to the database and who did it also what did they change. This will also track any files or information the user requested or gained access to. |

**\***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 | Brittney Miller |  |
| 2.0 | 8/7/2024 | Moduel 3 | Brittney Miller | [Insert text.] |
| 3.0 | 8/11/2024 | Redid entire template | Brittney Miller | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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