**Green Pace Developer: Security Policy**



# 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 | This principle emphasizes the importance of thoroughly validating all input data received by a system to prevent malicious exploitation through techniques like injection attacks. By validating input data, developers ensure that only expected and safe values are processed, reducing the risk of security vulnerabilities. |
| 1. Heed Compiler Warnings | This principle advises developers to pay attention to warnings issued by compilers during the software development process. Compiler warnings often indicate potential issues in the code, including security vulnerabilities. Ignoring these warnings can lead to exploitable weaknesses in the final product. |
| 1. Architect and Design for Security Policies | This principle advocates for incorporating security policies into the architectural and design phases of software development. By considering security requirements from the outset, developers can create systems that are inherently more secure and resistant to potential threats. |
| 1. Keep It Simple | The principle of keeping it simple suggests that simplicity in design and implementation can contribute to improved security. Complex systems are often harder to secure and prone to overlooked vulnerabilities. Simplifying designs and minimizing unnecessary complexity can reduce the attack surface and make the system more manageable and secure. |
| 1. Default Deny | This principle involves adopting a default deny stance for system access. In other words, access to resources or functionalities should be denied by default unless explicitly allowed. By starting with a stance of denial and selectively granting permissions, developers can better control access and mitigate the risk of unauthorized actions. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege advocates for granting users or processes only the minimum level of access or permissions necessary to perform their tasks. By limiting access rights, developers can reduce the potential impact of security breaches and limit the scope of potential damage. |
| 1. Sanitize Data Sent to Other Systems | This principle underscores the importance of sanitizing or cleansing data before transmitting it to other systems or components. Sanitization helps ensure that data sent from one part of the system to another is free from potentially harmful or malicious content, reducing the risk of data corruption or exploitation. |
| 1. Practice Defense in Depth | Defense in depth involves implementing multiple layers of security defenses to protect against various types of threats. By employing redundant and overlapping security measures, developers can create a more resilient and robust defense posture, making it harder for attackers to penetrate the system. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques, such as testing and code review, play a crucial role in ensuring the security of software systems. By rigorously testing for vulnerabilities and conducting thorough code reviews, developers can identify and address security issues before they are deployed into production environments. |
| 1. Adopt a Secure Coding Standard | Finally, adopting a secure coding standard provides developers with guidelines and best practices for writing secure code. By adhering to established coding standards, developers can minimize the risk of introducing vulnerabilities and ensure consistency in security practices throughout the development process. |

### 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** | [DCL-051-CPP] | Do not declare or define a reserved identifier. |

| **Noncompliant Code** |
| --- |
| A common practice is to use a macro in a preprocessor conditional that guards against multiple inclusions of a header file. While this is a recommended practice, many programs use reserved names as the header guards. Such a name may clash with reserved names defined by the implementation of the C++ standard template library in its headers or with reserved names implicitly predefined by the compiler even when no C++ standard library header is included. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| This compliant solution avoids using leading or trailing underscores in the name of the header guard. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

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

| * Principle 4 (Keep It Simple): Avoiding the declaration or definition of reserved identifiers simplifies code and reduces the risk of conflicts or unexpected behavior, aligning with the standard's emphasis on clarity and simplicity. * Principle 9 (Use Effective Quality Assurance Techniques): Adhering to this standard helps prevent naming conflicts and ensures code stability, which can be verified through effective quality assurance practices. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **reserved-identifier** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreserved-id-macro -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++5003** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.1 | **MISRA.DEFINE.WRONGNAME** **MISRA.DEFINE.WRONGNAME.UNDERSCORE** **MISRA.UNDEF.WRONGNAME** **MISRA.UNDEF.WRONGNAME.UNDERSCORE** **MISRA.STDLIB.WRONGNAME** **MISRA.STDLIB.WRONGNAME.UNDERSCORE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **86 S, 218 S, 219 S, 580 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL51-a** **CERT\_CPP-DCL51-b** **CERT\_CPP-DCL51-c** **CERT\_CPP-DCL51-d** **CERT\_CPP-DCL51-e** **CERT\_CPP-DCL51-f** | Do not #define or #undef identifiers with names which start with underscore Do not redefine reserved words Do not #define nor #undef identifier 'defined' The names of standard library macros, objects and functions shall not be reused The names of standard library macros, objects and functions shall not be reused (C90) The names of standard library macros, objects and functions shall not be reused (C99) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: DCL51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl51cpp.html) | Checks for redefinitions of reserved identifiers (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.30 | [**V1059**](https://pvs-studio.com/en/docs/warnings/v1059/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **reserved-identifier** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**978**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-978) |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [DCL-055-CPP] | Avoid information leakage when passing a class object across a trust boundary. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example runs in kernel space and copies data from arg to user space. However, padding bits may be used within the object, for example, to ensure the proper alignment of class data members. These padding bits may contain sensitive information that may then be leaked when the data is copied to user space, regardless of how the data is copied. |
| #include <cstddef>    struct test {    int a;    char b;    int c;  };    // Safely copy bytes to user space  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {    test arg{1, 2, 3};    copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

| **Compliant Code** |
| --- |
| This compliant solution serializes the structure data before copying it to an untrusted context. |
| #include <cstddef>  #include <cstring>    struct test {    int a;    char b;    int c;  };    // Safely copy bytes to user space.  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {    test arg{1, 2, 3};    // May be larger than strictly needed.    unsigned char buf[sizeof(arg)];    std::size\_t offset = 0;      std::memcpy(buf + offset, &arg.a, sizeof(arg.a));    offset += sizeof(arg.a);    std::memcpy(buf + offset, &arg.b, sizeof(arg.b));    offset += sizeof(arg.b);    std::memcpy(buf + offset, &arg.c, sizeof(arg.c));    offset += sizeof(arg.c);      copy\_to\_user(usr\_buf, buf, offset /\* size of info copied \*/);  } |

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

| * Principle 6 (Adhere to the Principle of Least Privilege): Avoiding information leakage when passing class objects across trust boundaries restricts access to only necessary data, aligning with the standard's goal of minimizing access privileges. * Principle 8 (Practice Defense in Depth): Preventing information leakage enhances the application's defense in depth strategy by reducing the exposure of sensitive data to untrusted components. * Principle 9 (Use Effective Quality Assurance Techniques): Following this standard helps detect and rectify information leakage issues during quality assurance testing, ensuring the overall security of the codebase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL55** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **MISC.PADDING.POTB** | Padding Passed Across a Trust Boundary |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **DF4941, DF4942, DF4943** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL55-a** | A pointer to a structure should not be passed to a function that can copy data to the user space |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | [CERT C++: DCL55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl55cpp.html) | Checks for information leakage due to structure padding (rule partially covered) |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [EXP-052-CPP] | Do not rely on side effects in unevaluated operands. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the expression a++ is not evaluated. |
| #include <iostream>  void f() {    int a = 14;    int b = sizeof(a++);    std::cout << a << ", " << b << std::endl;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the variable a is incremented outside of the sizeof operator. |
| #include <iostream>  void f() {    int a = 14;    int b = sizeof(a);    ++a;    std::cout << a << ", " << b << std::endl;  } |

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

| * **Principle 4 (Keep It Simple):** Dereferencing null pointers can introduce unnecessary complexity and lead to undefined behavior. Adhering to this standard simplifies code by avoiding such risky operations. * **Principle 8 (Practice Defense in Depth):** Dereferencing null pointers can result in segmentation faults or other runtime errors, potentially exposing security vulnerabilities. By avoiding this practice, developers strengthen the application's defense in depth strategy. * **Principle 9 (Use Effective Quality Assurance Techniques):** Preventing null pointer dereference is crucial for maintaining code quality and reliability. By following this standard, developers can detect and rectify null pointer-related issues during quality assurance testing, enhancing the overall robustness of the codebase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **sizeof** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-EXP52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wunevaluated-expression |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.SE.SIZEOF** | Side Effects in sizeof |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++3240, C++3241** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **MISRA.SIZEOF.SIDE\_EFFECT** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **54 S, 133 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-EXP52-a** **CERT\_CPP-EXP52-b** **CERT\_CPP-EXP52-c CERT\_CPP-EXP52-d CERT\_CPP-EXP52-e** | The operand of the sizeof operator shall not contain any expression which has side effects Object designated by a volatile lvalue should not be accessed in the operand of the sizeof operator The function call that causes the side effect shall not be the operand of the sizeof operator The operand of the 'typeid' operator shall not contain any expression that has side effects The operand of the 'typeid' operator shall not contain a function call that causes side effects |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: EXP52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp52cpp.html) | Checks for logical operator operand with side effects |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **sizeof** | Partially checked |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [EXP-058-CPP] | Pass an object of the correct type to va\_start. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the object passed to va\_start() will undergo a default argument promotion, which results in undefined behavior. |
| #include <cstdarg>    extern "C" void f(float a, ...) {    va\_list list;    va\_start(list, a);    // ...    va\_end(list);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, f() accepts a double instead of a float. |
| #include <cstdarg>    extern "C" void f(double a, ...) {    va\_list list;    va\_start(list, a);    // ...    va\_end(list);  } |

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

| * Principle 4 (Keep It Simple): Passing an object of the correct type to va\_start improves code clarity and reduces the risk of undefined behavior, aligning with the standard's emphasis on writing clear and straightforward code. * Principle 9 (Use Effective Quality Assurance Techniques): Following this standard helps ensure the correct usage of va\_start, which can be verified through effective quality assurance practices. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wvarargs | Does not catch the violation in the third noncompliant code example (it is conditionally supported by [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-Clang)) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **BADMACRO.STDARG\_H** | Use of <stdarg.h> Feature |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++3852, C++3853** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **CERT.VA\_START.TYPE** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-EXP58-a** | Use macros for variable arguments correctly |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: EXP58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp58cpp.html) | Checks for incorrect data types for second argument of va\_start (rule fully covered) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [INT-050-CPP] | Do not cast to an out-of-range enumeration value. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two's complement system, the valid range of values that can be represented by EnumType are [0..3], so if a value outside of that range were passed to f(), the cast to EnumType would result in an unspecified value, and using that value within the if statement results in [unspecified behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-unspecifiedbehavior). |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restricting the converted value to one for which there is a specific enumerator value. |
| 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.**

| * Principle 4 (Keep It Simple): Avoiding casts to out-of-range enumeration values simplifies code and reduces the risk of unintended behavior or undefined results, aligning with the standard's emphasis on clarity and simplicity. * Principle 9 (Use Effective Quality Assurance Techniques): Adhering to this standard helps prevent errors caused by invalid enumeration casts, which can be verified through effective quality assurance practices. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **cast-integer-to-enum** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-INT50** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters Value  Cast Alters Value |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++3013** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-INT50-a** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.30 | [**V1016**](https://pvs-studio.com/en/docs/warnings/v1016/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **cast-integer-to-enum** | Partially checked |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: INT50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcint50cpp.html) | Checks for casting to out-of-range enumeration value (rule fully covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STR-050-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

| **Noncompliant Code** |
| --- |
| To solve this problem, it may be tempting to use the std::ios\_base::width() method, but there still is a trap, as shown in this noncompliant code example. |
| #include <iostream>    void f() {    char bufOne[12];    char bufTwo[12];    std::cin.width(12);    std::cin >> bufOne;    std::cin >> bufTwo;  } |

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string instead of a bounded array, as in this compliant solution. |
| #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.**

| * Principle 4 (Keep It Simple): Guaranteeing sufficient storage for strings simplifies code and reduces the risk of buffer overflows or memory corruption, aligning with the standard's emphasis on clarity and simplicity. * Principle 8 (Practice Defense in Depth): Proper string storage management enhances the application's defense in depth strategy by mitigating the risk of memory-related vulnerabilities. * Principle 9 (Use Effective Quality Assurance Techniques): Following this standard helps detect and rectify string storage issues during quality assurance testing, ensuring the overall reliability of the codebase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++5216**  **DF2835, DF2836, DF2839,** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stream-input-char-array** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3519**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3519) |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STR-051-CPP] | Do not attempt to create a std::string from a null pointer. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, 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.**

| * Principle 4 (Keep It Simple): Avoiding attempts to create std::string from a null pointer simplifies code and reduces the risk of undefined behavior, aligning with the standard's emphasis on clarity and simplicity. * Principle 9 (Use Effective Quality Assurance Techniques): Adhering to this standard helps prevent errors caused by invalid string creation, which can be verified through effective quality assurance practices. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **assert\_failure** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **DF4770, DF4771, DF4772, DF4773, DF4774** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2024.1 | **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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023b | [CERT C++: STR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule partially covered). |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [FIO-050-CPP] | Do not alternately input and output from a file stream without an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example appends data to the end of a file and then reads from the same file. However, because there is no intervening positioning call between the formatted output and input calls, the behavior is [undefined](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";    std::string str;    file >> str;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::basic\_istream<T>::seekg() function is called between the output and input, eliminating the [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";      std::string str;    file.seekg(0, std::ios::beg);    file >> str;  } |

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

| * Principle 4 (Keep It Simple): Avoiding alternate input and output simplifies file stream handling and reduces the risk of unexpected behavior, aligning with the standard's emphasis on clarity and simplicity. * Principle 8 (Practice Defense in Depth): Proper file stream management enhances the application's defense in depth strategy by mitigating the risk of file corruption or data loss. * Principle 9 (Use Effective Quality Assurance Techniques): Following this standard helps detect and rectify file stream handling issues during quality assurance testing, ensuring the overall reliability of the codebase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-FIO50** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **IO.IOWOP**  **IO.OIWOP** | Input After Output Without Positioning  Output After Input Without Positioning |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **DF4711, DF4712, DF4713** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-FIO50-a** | Do not alternately input and output from a stream without an intervening flush or positioning call |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: FIO50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio50cpp.html) | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [FIO-051-CPP] | Close files when they are no longer needed. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed. |
| #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** |
| --- |
| In this compliant solution, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| #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.**

| * Principle 4 (Keep It Simple): Closing files when no longer needed simplifies file handling and reduces the risk of resource leaks or file corruption, aligning with the standard's emphasis on clarity and simplicity. * Principle 8 (Practice Defense in Depth): Proper file closure enhances the application's defense in depth strategy by mitigating the risk of unauthorized access or resource exhaustion. * Principle 9 (Use Effective Quality Assurance Techniques): Adhering to this standard helps detect and rectify file handling issues during quality assurance testing, ensuring the overall reliability of the codebase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **DF4786, DF4787, DF4788** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **RH.LEAK** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [ERR-050-CPP] | Do not abruptly terminate the program. |

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

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

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

| * Principle 4 (Keep It Simple): Avoiding abrupt program termination simplifies error handling and reduces the risk of unexpected behavior or data loss, aligning with the standard's emphasis on clarity and simplicity. * Principle 8 (Practice Defense in Depth): Proper error handling enhances the application's defense in depth strategy by mitigating the risk of system instability or exploitation. * Principle 9 (Use Effective Quality Assurance Techniques): Following this standard helps detect and rectify error handling issues during quality assurance testing, ensuring the overall reliability of the codebase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stdlib-use** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++5014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **MISRA.TERMINATE** **CERT.ERR.ABRUPT\_TERM** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **122 S** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR50-a** **CERT\_CPP-ERR50-b** **CERT\_CPP-ERR50-c** **CERT\_CPP-ERR50-d** **CERT\_CPP-ERR50-e** **CERT\_CPP-ERR50-f** **CERT\_CPP-ERR50-g** **CERT\_CPP-ERR50-h** **CERT\_CPP-ERR50-i** **CERT\_CPP-ERR50-j** **CERT\_CPP-ERR50-k** **CERT\_CPP-ERR50-l** **CERT\_CPP-ERR50-m CERT\_CPP-ERR50-n** | The execution of a function registered with 'std::atexit()' or 'std::at\_quick\_exit()' should not exit via an exception Never allow an exception to be thrown from a destructor, deallocation, and swap Do not throw from within destructor There should be at least one exception handler to catch all otherwise unhandled exceptions An empty throw (throw;) shall only be used in the compound-statement of a catch handler Exceptions shall be raised only after start-up and before termination of the program Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) Function called in global or namespace scope shall not throw unhandled exceptions Always catch exceptions Properly define exit handlers The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used Avoid throwing exceptions from functions that are declared not to throw The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023b | [CERT C++: ERR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr50cpp.html) | Checks for implicit call to terminate() function (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.30 | [**V667**](https://pvs-studio.com/en/docs/warnings/v667/)**,** [**V2014**](https://pvs-studio.com/en/docs/warnings/v2014/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stdlib-use** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S990**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-990) |  |

### 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.

**Assess and Plan**: You can automate the evaluation of your current security protocols and identify areas for improvement. This can be achieved by integrating automated tools that can assess regulatory changes, threat landscapes, and evolving technologies, ensuring your systems are always up-to-date with the latest standards.

**Verify and Test**: Enhancing your automated testing frameworks to include security testing based on policy standards can help ensure that your applications are secure. Additionally, integrating vulnerability scanning tools into your testing phase can help identify and address potential security threats early in the development process. Recommend utilizing an automated approach to gathering standards, batching specific tools, and testing codebases.

**Monitor and Detect**: By enhancing your monitoring solutions with AI/ML for real-time anomaly detection, you can ensure that your systems are always monitored for potential security threats. Automating log collections and ensuring they are analyzed for security threats in real-time can help you quickly identify and address any potential security issues.

### 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 |
| --- | --- | --- | --- | --- | --- |
| DCL51-CPP | Low | Unlikely | Low | **P3** | **L3** |
| DCL55-CPP | Low | Unlikely | High | **P1** | **L3** |
| EXP52-CPP | Low | Unlikely | Low | **P3** | **L3** |
| EXP58-CPP | Medium | Unlikely | Medium | **P4** | **L3** |
| INT50-CPP | Medium | Unlikely | Medium | **P4** | **L3** |
| STR50-CPP | High | Likely | Medium | **P18** | **L1** |
| STR51-CPP | High | Likely | Medium | **P18** | **L1** |
| FIO50-CPP | Low | Likely | Medium | **P6** | **L2** |
| FIO51-CPP | Medium | Unlikely | Medium | **P4** | **L3** |
| ERR50-CPP | Low | Probable | 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 | This policy mandates that all sensitive data stored on our systems must be encrypted. This ensures that even if physical or virtual drives are compromised, the data they contain is unreadable. In practice, this could involve using disk encryption technologies like BitLocker or FileVault, or database-level encryption for data at rest. |
| Encryption in flight | This policy requires that all data sent over networks, whether internal or external, must be encrypted. This prevents unauthorized interception and access to sensitive data during transmission. In practice, this could involve using protocols like HTTPS for web traffic or TLS for other types of network traffic. |
| Encryption in use | This policy stipulates that data must be encrypted even when being used or processed. This ensures that data remains secure even if the system processing it is compromised. In practice, this could involve techniques like homomorphic encryption, which allows computations to be performed on encrypted data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This policy requires users to prove their identity before accessing our systems. In practice, this could involve using usernames and passwords, two-factor authentication, or biometric authentication. This ensures that only legitimate users can log in. |
| Authorization | This policy dictates what resources and services a user can access once they are authenticated. In practice, this could involve setting user roles and permissions, ensuring users can only access the resources necessary for their role. |
| Accounting | This policy mandates the tracking of user activities on our systems. In practice, this could involve logging user actions like logins, database changes, new user additions, changes in access levels, and files accessed by users. This creates an audit trail that can be used for troubleshooting, identifying misuse, and ensuring compliance with regulations. |

**\***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 | 4/21/2024 | Project One Submission | Kyle Blan |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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