**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 should be thoroughly checked for proper format, and length before being used by the system to prevent malicious activity like SQL injection, buffer overflows, numeric overflows, and other vulnerabilities. |
| 1. Heed Compiler Warnings | Often indicate potential bugs, undefined behavior, and other potential issues in code. Developers must treat warnings as errors and work to resolve them by properly updating code to prevent these vulnerabilities. |
| 1. Architect and Design for Security Policies | Every project begins with the architecture and design decisions. Developers need to consider how to create a safe and secure environment from the beginning to avoid having to redesign later. This may include creating secure users to create a hierarchy to allow differences in abilities. (IE, allowing admin users higher privileges than a normal user) |
| 1. Keep It Simple | Keeping the system design simple and small will help prevent overcomplicating the system and make it easier to test and fix any issues. |
| 1. Default Deny | Prevent any access or operations from a user unless explicitly permitted. This prevents unauthorized users from using the system by default, and will increase security |
| 1. Adhere to the Principle of Least Privilege | Users and operations should only be granted with minimum permissions to perform so that it prevents unauthorized users from getting access to restricted and sensitive data. |
| 1. Sanitize Data Sent to Other Systems | Checking any data passed between systems or components to verify data is valid. Improper data can lead to SQL Injections and other security vulnerabilities. |
| 1. Practice Defense in Depth | Using multiple layers of security can help mitigate attacks. One layer may be compromised but another layer can still protect the system. |
| 1. Use Effective Quality Assurance Techniques | Unit testing, penetration testing, security reviews, and other techniques can provide insight into the security of the system and help identify insecurities or other issues that can then be addressed by the developers. |
| 1. Adopt a Secure Coding Standard | Create an appropriate coding standard for the coding language and platform of choice and train developers to adhere to it to create a secure and functional system. |

### 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 the one-definition rule |

Reference: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule>

Rationale: Standard should be met when creating classes or structures.

| **Noncompliant Code** |
| --- |
| The following is an example of noncompliant code for this standard. The code creates two different definitions of a class which violates the rule. |
| // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |

| **Compliant Code** |
| --- |
| The following is an example of compliant code for this standard. The code a header file to introduce the object. |
| // S.h  **struct** S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

| **Principles(s):** 2 – Heed Compiler Warnings – Breaking the one-definition rule can lead to undefined behavior. Undefined behavior can cause bugs and create potential vulnerabilities that could be exploited.  4 – Keep it simple – The one-definition rule allows for clean reusable code by explicitly defining a function or variable in the program or can be defined using the standard library or user defined library.  10 – Adopt a Secure Coding Standard – The one-definition rule is an appropriate standard to implement a secure and functional program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Type-compatibilty  Definition-duplicate  Undefined-extern  Undefined-extern-pure-vertual  External-file-spreading  Type-file-spreading | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++ -DCL60 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | LANG.STRUCT.DEF.FDH  LANG.STRUCT.DEF.ODH | Function defined in header file  Object defined in header file |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | C++1067, C++1509, C++1510 |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 286 S, 287 S | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-DCL60-a | The One Definition Rule shall not be violated |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | CERT C++: DCL60-CPP | Checks for inline constraints not respected (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | Type-compatibility  Definition-duplicate  Undefined-extern  Undefined-extern-pure-vertual  External-file-spreading  Type-file-spreading | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not read uninitialized memory |

Reference: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory>

Rationale: Used when accessing a variable

| **Noncompliant Code** |
| --- |
| The following is an example of non-compliant code for this standard. An uninitialized variable is used resulting in undefined behavior |
| #include <iostream>    **void** f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| The following is an example of compliant code for this standard. The initialized variable is used resulting in expected behavior |
| #include <iostream>    **void** f() {  **int** i = 0;    std::cout << i;  } |

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

| **Principles(s):** 1 – Validate input data – Use the correct datatype for the user input  2 – Heed Compiler Warnings – The compiler may warn of variables that are uninitialized  4 – Keep it Simple – Always initialize the variable before using that variable  10 – Adopt a Secure Coding Standard – Any variables need to be initialized with a value for its datatype |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Uninitialized-read | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wuninitialized  Clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from the heap-allocated memory |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local  Uninitialized variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | UNINIT.CTOR.MIGHT  UNINIT.CTOR.MUST  UNINIT.HEAP.MIGHT  UNINIT.HEAP.MUST  UNINIT.STACK.ARRAY.MIGHT  UNINIT.STACK.ARRAY.MUST  UNINIT.STACK.ARRAY.PARTIAL.MUST  UNINIT.STACK.MIGHT  UNINIT.STACK.MUST |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially Implemented |
| [Parasoft C/C++ test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-EXP53-a | Avoiduse before inittialization |
| [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) | R2024b | CERT C++: EXP53-CPP | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.38 | V546, V573, V614, V670, V679, V730, V788, V1007, V1050 |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | Uninitialized-read | Partially checked |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | C54  C55  C56  C57  C58  C59 C60  C61  C62  C63 | Fully implemented |

#### 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 |

Reference: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer>

Rationale: Used when referencing pointer variables.

| **Noncompliant Code** |
| --- |
| The following is an example of non-compliant code for this standard. 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 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** |
| --- |
| The following is an example of compliant code for this standard. The results from the call to std::getenv() are checked for null before the std::string object is contructed. |
| #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 Warning – Trying to create an std::string variable from a null pointer would create undefined behavior. The compiler should warn for a nullptr exception.  10 – Adopt a Secure Coding Standard – Correctly initialize a string to avoid nullptrs |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | 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/cplusplus/CodeSonar) | 9.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | DF4770, DF4771, DF4772, DF4773, DF4774 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | 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) | 2024.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: STR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr51cpp.html) | Checks for string operations on null pointer (rule partially covered) |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | shiftTooManyBits | Fully implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Prevent SQL Injection |

Reference: <https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection>

Rationale: Used when processing SQL statements.

| **Noncompliant Code** |
| --- |
| The following is an example of non-compliant code for this standard. Uses unvalidated input which may allow malicious activity |
| #include <string>  std::string username;  std::string password;  std::cin >> username;  std::cin >> password;  std::string sql = “SELECT \* FROM users WHERE username=” + username; |

| **Compliant Code** |
| --- |
| The following is an example of compliant code for this standard. Uses validated input which prevents malicious activity |
| PreparedStatement pStmt = PreparedStatement();  std::cin >> username;  std::cin >> userpassword;  sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;”, username,  userpassword}; |

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

| **Principles(s):** 1 – Validate Input Data – Check input to prevent unwanted data that may cause issues  4 – Keep it simple – Use secure coding like a prepared statement to limit the ability for an SQL Injection  5 – Default Deny – Deny suspicious activity  6 – Adhere to the principle of least privilege - Limit access to data and processes to limit breaches to protect data  7 – Sanitize Data sent to other systems – Check for unused functions or calls made out of context to prevent access  8 – Practice Defense in depth – Input validation, prepared statements, and sanitizing data is a security strategy that fits defense in depth  9 – Use effective quality assurance techniques – Unit testing for SQL Injection is an effective approach to prevent SQL Injection from occurring  10 – Adopt a secure coding standard – SQL Injections is a common security vulnerability that is exploited. Preventing SQL attacks in your code is a must. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P1 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2025.6.0 | SQLI | Detects SQL injection vulnerabilities by analyzing data flow from untrusted sources to SQL queries |
| Fortify | 25.2.0 | SQL\_INJECTION\_Persistence  SQL\_INJECTION | Identifies SQL injection flaws by tracking tainted data from its source to its use in a database query |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | SV.DATA.DB  SV.SQL | Flags code where user input is used to construct SQL statements, which could lead to injection attacks |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 9.9 | S2077  S3649 | Provides rules to ensure SQL queries are built safely, preventing common injection vulnerabilities in C++ |

#### Coding Standard 5

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

Reference: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory>

Rationale: Used when accessing a variable.

| **Noncompliant Code** |
| --- |
| The following is an example of non-compliant code for this standard. It attempts to access dereferenced value |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| The following is an example of compliant code for this standard. The memory is not deallocated until it is 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 – The compiler will warn of attempting to use freed memory  5 – Default Deny – Deny access to prevent access to data during run time errors like overflows  10 – Adopt a Secure Coding Standard – Not accessing freed memory will protect memory |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Dangling\_pointer\_use |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-MEM50 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | Clang-analyzer-cplusplus.NewDelete  Clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | ALLOC.UAF | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallogcated more than once or read/written to the target of the freed pointer |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | C++4303, C++4304 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2025.2 | UFM.DEREF.MIGHT  UFM.DEREF.MUST  UFM.FFM.MIGHT  UFM.FFM.MUST  UFM.RETURN.MIGHT  UFM.RETURN.MUST  UFM.USE.MIGHT  UFM.USE.MUST |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 483 S, 484 S | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-MEM50-a | Do not use resources that have been 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) | R2024b | CERT C++: MEM50-CPP | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.38 | [V586](https://pvs-studio.com/en/docs/warnings/v586/), [V774](https://pvs-studio.com/en/docs/warnings/v774/) |  |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | CPP\_12  CPP\_14  CPP\_15 | Fully implemented |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Splint) | 5.0 |  |  |

#### 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 |

Reference: <https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression>

Rationale: Used when testing a value of a constant expression

| **Noncompliant Code** |
| --- |
| The following is an example of non-compliant code for this standard. It uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
| #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** |
| --- |
| The following is an example of compliant code for this standard. It uses a preprocessor conditional statement. |
| **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):** 9 – Use Effective Quality Assurance Techniques – Using the static assertion as a way to test is part of quality assurance and makes sure the value is producing the expected value  10 – Adopt a Secure Coding Standard – Assertions are part of unit testing which should be a standard practice to test the code |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL03 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | Misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 9.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Cout detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | CC2.DCL03 | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Fully implemented |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/c/Security+Reviewer+-+Static+Reviewer) | 6.02 | C13  C14  C15  C52  C129  C130  C132  C133  C154  C155 |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Do not abruptly terminate the program |

Reference: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR50-CPP.+Do+not+abruptly+terminate+the+program>

Rationale: Properly handle error messaging to handheld program failures.

| **Noncompliant Code** |
| --- |
| The following is an example of non-compliant code for this standard. 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** |
| --- |
| The following is an example of compliant code for this standard. 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.**

| **Principles(s):** 1 – Validate Input Data – Exception handling is key to making sure input is valid and or handled correctly if the user inputs unexpected or incorrect input values  2 – Heed Compiler – Compiler may warn of an exception that needs to be handled  4 – Keep It Simple – Handling exceptions will keep the code simple  7 – Sanitize Data Sent to Other Systems – Handling exceptions when receiving or sending data is key to making sure the data is valid and if not valid it is handled without creating issues or vulnerabilities  9 – Use Effective Quality Assurance Techniques – Testing that exceptions are thrown as expected and caught is a good way to test the programs functionality  10 – Adopt a Secure Coding Standard – Exception handling is a secure coding standard that should be adopted so that the program is secure if there are exceptions thrown during processing |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | 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) | 9.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | C++5014 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | 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) | 2024.2 | 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 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) | R2024b | CERT C++: ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.38 | [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) |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | STD-008-CPP | Do not use pointer-to-member operators to access nonexistent members |

Reference: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP55-CPP.+Do+not+use+pointer-to-member+operators+to+access+nonexistent+members>

Rationale: Used when accessing member objects

| **Noncompliant Code** |
| --- |
| The following is an example of non-compliant code for this standard. A pointer-to-member object is obtained from D::g but is then upcast to be a B::\*. When called on an object whose dynamic type is D, the pointer-to-member call is well defined. However, the dynamic type of the underlying object is B, which results in undefined behavior. |
| **struct** B {  **virtual** ~B() = **default**;  };    **struct** D : B {  **virtual** ~D() = **default**;  **virtual** **void** g() { /\* ... \*/ }  };    **void** f() {    B \*b = **new** B;      // ...    **void** (B::\*gptr)() = **static\_cast**<**void**(B::\*)()>(&D::g);    (b->\*gptr)();  **delete** b;  } |

| **Compliant Code** |
| --- |
| The following is an example of compliant code for this standard. The upcast is removed, rendering the initial code ill-formed and emphasizing the underlying problem that B::g() does not exist. This compliant solution assumes that the programmer’s intention was to use the correct dynamic type for the underlying object. |
| **struct** B {  **virtual** ~B() = **default**;  };    **struct** D : B {  **virtual** ~D() = **default**;  **virtual** **void** g() { /\* ... \*/ }  };    **void** f() {    B \*b = **new** D; // Corrected the dynamic object type.      // ...  **void** (D::\*gptr)() = &D::g; // Moved static\_cast to the next line.    (**static\_cast**<D \*>(b)->\*gptr)();  **delete** b;  } |

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

| **Principles(s):** 9 – Use Effective Quality Assurance Techniques – Test members before using them to avoid issues with nullptr errors  10 – Adopt a Secure Coding Standard – Object Oriented Programming is an important standard to use for secure coding |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | Overflow\_upon\_dereference  Invalid\_function\_pointer |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-OOP55 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | LANG.MEM.UVAR | Uninitialized Variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | DF2810, DF2811, DF2812, DF2813, DF2814 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2025.2 | CERT.OOP.PTR\_MEMBER.NO\_MEMBER |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | CERT\_CPP-OOP55-a | A cast shall not convert a pointer to a function to any other pointer type, including a pointer to function type |
| [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) | R2024b | [CERT C++: OOP55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop55cpp.html) | Check for pointers to member accessing non-existent class members (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | STD-009-CPP | Do not cast to an out-of-range enumeration value |

Reference: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/INT50-CPP.+Do+not+cast+to+an+out-of-range+enumeration+value>

Rationale: Used when accessing enumeration values

| **Noncompliant Code** |
| --- |
| The following is an example of non-compliant code for this standard. This non-compliant 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. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {    EnumType enumVar = **static\_cast**<EnumType>(intVar);    **if** (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| The following is an example of compliant code for this standard. It 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.**

| **Principles(s):** 1 – Validate Input Data – Validate the datatype is big enough to handle the value  2 – Heed Compiler Warnings – Compiler may warn against out of range values |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | High | 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) | 9.1p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | C++3013 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | 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) | 7.38 | [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) | R2024b | [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 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | STD-010-CPP | Close files when they are no longer needed |

Reference: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed>

Rationale: Used to properly close files after use.

| **Noncompliant Code** |
| --- |
| The following is an example of non-compliant code for this standard. 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** |
| --- |
| The following is an example of compliant code for this standard. It uses an std::fstream::close() 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.**

| **Principles(s):** 5 – Default Deny – Closing the file after use protects against unwanted access  6 – Adhere to the Principle of Least Privilege – Only open and close files with minimal privileges  8 – Practice Defense in Depth – Closing the file is a layer of defense |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | ALLOC.LEAK | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2025.2 | DF4786, DF4787, DF4788 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2025.2 | RH.LEAK |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | 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) | R2024b | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | C80 | Fully Implemented |

### 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.

Adding automation for security to create the DevSecOps process is important. The need to integrate security is to ensure security is included in each phase. Automating will help efficiently add security to each phase and prevent issues later in the cycle. During the Design phase there will now be integrated tools into the IDE’s to automate the policies. Tools like Coverity, Klocwork, Clang, and others will scan code now during each new iteration of the code to find vulnerabilities and errors so the developer can fix it right away. For the Building and Test phases, the continued use of static application security testing tools like Coverity will be part of the acceptance criteria. Testing tools will scan the builds as they are submitted. Testing will include unit tests to help test values and find undesired behavior.

For the production phases, continuous monitoring and analysis will be key. Policies like deny default and least privileges will be a layer of the defense in depth approach. We will also use the logging and monitoring to ensure activity is monitored. Finding unusual activity will help us respond fast and be more efficient in resolving 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 |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Probable | Low | P12 | L1 |
| STD-003-CPP | High | Likely | Low | P18 | L1 |
| STD-004-CPP | High | Probable | Medium | P1 | L1 |
| STD-005-CPP | High | Likely | Medium | P9 | L2 |
| STD-006-CPP | Low | Unlikely | High | P3 | L3 |
| STD-007-CPP | Low | Probable | High | P2 | L3 |
| STD-008-CPP | High | Probable | Medium | P6 | L2 |
| STD-009-CPP | Medium | Unlikely | High | P4 | L3 |
| STD-010-CPP | Medium | Unlikely | High | P2 | 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 at rest when it is stored on a hard drive or database. Encrypting this data is important because this data is often the most important for programs. This is often the most valuable data hackers can get, which is why it needs to be secured and encrypted. |
| Encryption in flight | Data in flight is when it is being transmitted from one computer to another, or across a network. Because it is in transit it makes the data susceptible to interception attacks and get stolen. To protect data in transit the data needs to be encrypted by a secure algorithm and only accessible with a key that will unencrypt it. |
| Encryption in use | Data in use is what the data is actively being viewed or used on the program. Encrypting this data means making sure the data is only accessible by those with the correct permissions to access it. Using the least privileges that a user need will protect the data from unwanted access if users are attacked. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Proper authentication will protect users from attacks. Using multifactor authentication and other protocols like CAPTCHA will prevent unauthenticated access and protect data. |
| Authorization | Only allowing the least privileges to users will make sure only certain data is accessed by users. This helps prevent data loss if a user is attacked and accessed by a hacker. |
| Accounting | Logging user activity to monitor for unusual activity is a keyway to be secure. Finding unusual activity can lead to quicker response times to attacks which will hopefully prevent the least amount of data to be retrieved by any attacks. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 07/19/2025 | Milestone | Austin Donaubauer |  |
| 1.2 | 08/08/2025 | Project One | Austin Donaubauer |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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