**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

# Charles Adkins

## Contents

[Green Pace Secure Development Policy 0](#_Toc164009957)

[Charles Adkins 0](#_Toc164009958)

[Contents 1](#_Toc164009959)

[Overview 3](#_Toc164009960)

[Purpose 3](#_Toc164009961)

[Scope 3](#_Toc164009962)

[Module Three Milestone 3](#_Toc164009963)

[Ten Core Security Principles 3](#_Toc164009964)

[C/C++ Ten Coding Standards 4](#_Toc164009965)

[Coding Standard 1 5](#_Toc164009966)

[Coding Standard 2 7](#_Toc164009967)

[Coding Standard 3 9](#_Toc164009968)

[Coding Standard 4 11](#_Toc164009969)

[Coding Standard 5 13](#_Toc164009970)

[Coding Standard 6 15](#_Toc164009971)

[Coding Standard 7 17](#_Toc164009972)

[Coding Standard 8 20](#_Toc164009973)

[Coding Standard 9 22](#_Toc164009974)

[Coding Standard 10 24](#_Toc164009975)

[Defense-in-Depth Illustration 26](#_Toc164009976)

[Project One 26](#_Toc164009977)

[Revise the C/C++ Standards 26](#_Toc164009978)

[Risk Assessment 26](#_Toc164009979)

[Automated Detection 26](#_Toc164009980)

[Automation 26](#_Toc164009981)

[Summary of Risk Assessments 27](#_Toc164009982)

[Create Policies for Encryption and Triple A 27](#_Toc164009983)

[Map the Principles 28](#_Toc164009984)

[Audit Controls and Management 30](#_Toc164009985)

[Enforcement 30](#_Toc164009986)

[Exceptions Process 30](#_Toc164009987)

[Distribution 31](#_Toc164009988)

[Policy Change Control 31](#_Toc164009989)

[Policy Version History 31](#_Toc164009990)

[Appendix A Lookups 31](#_Toc164009991)

[Approved C/C++ Language Acronyms 31](#_Toc164009992)

## 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 | Validates input data to ensure it meets the criteria of the system and to prevent malicious efforts. |
| 1. Heed Compiler Warnings | Notifies of errors occurring. This communicates required modifications that allows for proper execution of the program. |
| 1. Architect and Design for Security Policies | Architecture and design must consider executable requirements, security policies and layering of the system, which reinforces access level controls and privileges. |
| 1. Keep It Simple | Simplicity ensures the system doesn’t over-complicate its functioning or cause unnecessary errors. |
| 1. Default Deny | By default, access is denied, and access is permitted through the conditions of the protection scheme used. |
| 1. Adhere to the Principle of Least Privilege | Grants minimal required privileges needed to complete the job, and elevated privileges should be used as minimally as possible. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data before passing the data to other systems checks these potential issues prior to invoking these systems, such as functions made out of context such as SQL injection attacks. |
| 1. Practice Defense in Depth | Having multiple layers of defense bolsters the likelihood of handling threats or attacks from various sources and vectors. |
| 1. Use Effective Quality Assurance Techniques | Testing and code base audits are part of an effective QA program to ensure bugs and vulnerabilities are patched and corrected. |
| 1. Adopt a Secure Coding Standard | Apply coding standards in the system ensures readability and reproducibility. |

### 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] | Each component has a single definition. |

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

| **Noncompliant Code** |
| --- |
| Two different translation units define a class of the same name with differing definitions. |
| // x.cpp  struct A {  **int** b;  };    // y.cpp  class A {  public:  **int** b;  }; |

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

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C++ test | 2023.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| LDRA | 9.7.1 | 286 S, 287 S | Fully Implemented |
| Axivion Bauhaus Suite | 7.2 | CertC++-DCL60 |  |

#### Coding Standard 2

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

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

| **Noncompliant Code** |
| --- |
| Uninitialized local variable is evaluated as part of an expression to print its value, resulting in undefined behavior. |
| #include <iostream>    void x() {  **int** y;    std::cout << y;  } |

| **Compliant Code** |
| --- |
| Object is initialized prior to printing its value. |
| #include <iostream>    void x() {  **int** y = 0;    std::cout << y;  } |

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

| **Principles(s):**  1: ValidateInput Data  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Astrée | 22.1 | uninitialized-read | Partially checked |
| Polyspace Bug Finder | R2023b | CERT C++: EXP53-CPP | Non-initialized variable  Non-initialized pointer |

#### 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 |

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

| **Noncompliant Code** |
| --- |
| 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 x() {    std::string tmp(std::getenv("ABC"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| The results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    void x() {    const **char** \*tmpPtrVal = std::**getenv**("ABC");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):**  2: Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Astrée | 22.1 | Assert\_failure |  |
| Helix QAC | 2024.1 | C++4770, C++4771, C++4772, C++4773, C++4774 |  |

#### Coding Standard 4

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

Java version source (for reference): <https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection>

| **Noncompliant Code** |
| --- |
| Without precautions, the untrusted data may maliciously alter the query. |
| Name = getUsername("username");  Pass = getPassword("userpassword");  sql = “SELECT \* FROM Users WHERE Name = " + Name + " AND Pass = " +  Pass + ” |

| **Compliant Code** |
| --- |
| The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures. |
| PreparedStatement pStmt = PreparedStatement();  std::cin >> Username;  std::cin >> Password;  sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;”, Username,  Password}; |

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

| **Principles(s):**  1: ValidateInput Data  7: Sanitize Data Sent to Other Systems  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Jtest | 2023.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| Fortify | 1.0 | SQL\_Injection\_\_Persistence  SQL\_Injection | Implemented |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |

#### Coding Standard 5

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

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

| **Noncompliant Code** |
| --- |
| a is dereferenced after it has been deallocated. If this access results in a write-after-free, this can be exploited to run arbitrary code with the permissions of the vulnerable process. |
| #include <new>    struct A {    void x();  };    void x() noexcept(false) {    A \*a = new a;    // ...    delete a;    // ...    a->x();  } |

| **Compliant Code** |
| --- |
| The dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct A {    void x();  };    void x() noexcept(false) {    A \*a = new A;    // ...    a->x();    delete a;  } |

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

| **Principles(s):**  2: Heed Compiler Warnings  5: Default Deny  6: Adhere to the Principle of Least Privilege |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_C-MEM30-a | Do not use resources that have been freed |
| Coverity | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Parasoft Insure++ |  |  | Runtime analysis |

#### Coding Standard 6

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

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

| **Noncompliant Code** |
| --- |
| 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** |
| --- |
| For constant expressions, a preprocessor conditional statement may be used. |
| struct timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA | 9.7.1 | 44 S | Fully Implemented |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| ECLAIR | 1.2 | CC2.DCL03 | Fully Implemented |

#### Coding Standard 7

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

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

| **Noncompliant Code** |
| --- |
| 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** |
| --- |
| 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):**  9: Use Effective Quality Assurance Techniques  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.1 | C++5014 |  |
| LDRA | 9.7.1 | 122 S | Enhanced Enforcement |
| Parasoft C/C++ test | 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 |
| RuleChecker | 22.1 | stdlib-use | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Use valid iterator ranges |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterator+ranges>

| **Noncompliant Code** |
| --- |
| On each iteration of its internal loop, std::for\_each() compares the first iterator (after incrementing it) with the second for equality; as long as they are not equal, it will continue to increment the first iterator. Incrementing the iterator representing the past-the-end element of the range results in undefined behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<**int**> &c) {    std::for\_each(c.end(), c.begin(), [](**int** i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| The iterator values passed to std::for\_each() are passed in the proper order. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<**int**> &c) {    std::for\_each(c.begin(), c.end(), [](**int** i) { std::cout << i; });  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.MEM.BO | Buffer overrun |
| Helix QAC | 2024.1 | C++3802 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-CTR53-a CERT\_CPP-CTR53-b | Do not use an iterator range that isn't really a range Do not compare iterators from different containers |
| Polyspace Bug Finder | R2023b | CERT C++: CTR53-CPP | Checks for invalid iterator range (rule partially covered). |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Write constructor member initializers in the canonical order |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP53-CPP.+Write+constructor+member+initializers+in+the+canonical+order>

| **Noncompliant Code** |
| --- |
| The member initializer list for C::C() attempts to initialize someVal first and then to initialize dependsOnSomeVal to a value dependent on someVal. Because the declaration order of the member variables does not match the member initializer order, attempting to read the value of someVal results in an unspecified value being stored into dependsOnSomeVal. |
| class C {  **int** dependsOnSomeVal;  **int** someVal;    public:    C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| Change the declaration order of the class member variables so that the dependency can be ordered properly in the constructor's member initializer list. |
| class C {  **int** someVal;  **int** dependsOnSomeVal;    public:    C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

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

| **Principles(s):**  4: Keep It Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA | 9.7.1 | 206 S | Fully implemented |
| Astrée | 20.1 | initializer-list-order | Fully checked |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-OOP53-a** | List members in an initialization list in the order in which they are declared |
| Polyspace Bug Finder | R2023b | CERT C++: OOP53-CPP | Checks for members not initialized in canonical order (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Do not access an object outside of its lifetime |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP54-CPP.+Do+not+access+an+object+outside+of+its+lifetime>

| **Noncompliant Code** |
| --- |
| A pointer to an object is used to call a non-static member function of the object prior to the beginning of the pointer's lifetime, resulting in undefined behavior. |
| struct S {    void mem\_fn();  };    void f() {    S \*s;    s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| Storage is obtained for the pointer prior to calling S::mem\_fn(). |
| struct S {    void mem\_fn();  };    void f() {    S \*s = new S;    s->mem\_fn();    delete s;  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wdangling-initializer-list | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |
| CodeSonar | 8.1p0 | IO.UAC  ALLOC.UAF | Use after close Use after free |
| Helix QAC | 2024.1 | C++4003, C++4026  DF2812, DF2813, DF2814, DF2930, DF2931, DF2932, DF2933, DF2934 |  |
| LDRA | 9.7.1 | 42 D, 53 D, 77 D, 1 J, 71 S, 565 S | Partially 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.

[Insert your written explanations here.]

### 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 | High | Low (3) | 3 |
| STD-002-CPP | High | Probable | Medium | High (12) | 1 |
| STD-003-CPP | High | Likely | Medium | High (18) | 1 |
| STD-004-CPP | High | Probable | Medium | High (12) | 1 |
| STD-005-CPP | High | Likely | Medium | High (18) | 1 |
| STD-006-CLG | Low | Unlikely | High | Low (1) | 3 |
| STD-007-CPP | Low | Probable | Medium | Low (4) | 3 |
| STD-008-CPP | High | Probable | High | Medium (6) | 2 |
| STD-008-CPP | Medium | Unlikely | Medium | Low (4) | 3 |
| STD-010-CPP | High | Probable | High | Medium (6) | 2 |

### 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 type protects stored data, which include hard drives, phones, computers, etc. Protection of this data can be done through encryption tools, disk encryption and security for mobile devices and computers. |
| Encryption in flight | This type protects data that is on the move. This can be through a network or moving outside of a network. Such as email encryption, DLP solutions, security firewalls and user authentication |
| Encryption in use | This type protects data that is created, modified or in-use. This can be accomplished by enforcing data control and protection, managing access controls and authentication. |

Source: <https://www.mimecast.com/blog/data-in-transit-vs-motion-vs-rest/>

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the act of confirming a users’ identity. This can include passwords, certifications, and biometric credentials. These forms of identification confirm that a person is who they claim to be. |
| Authorization | This outlines rights and privileges of a user, which determines what the user can and cannot access. This limits potential vulnerabilities by restricting access to data that is public, private, need-to-know or other classifications, as they relate to the specific users authorization level. |
| Accounting | Maintaining a log of activity that details actions and behavior with a system. This data is useful for monitoring purposes, but also for investigative efforts like forensic analysis. |

Source: <https://www.ccsinet.com/blog/aaa-identity-management/>

**\***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 | 03/31/2024 | Module 3 Milestone | Charles Adkins |  |
| 1.2 | 04/14/2024 | Module 6 Project One | Charles Adkins |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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