**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

# Thomas Bartlett

## 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 | Check the Input Data coming into the system from the user to make only proper input is used and invalid or harmful data is caught before it is used. |
| 1. Heed Compiler Warnings | These warnings exist to alert the programmer that there may be a potential error or some kind of issue with the way the code is written that could cause vulnerabilities to the system. Although the warning will not prevent the code from compiling, a programmer should still consider looking at the warnings to see if code correction is needed. |
| 1. Architect and Design for Security Policies | Take into consideration how a system is created and designed when implementing any security policies. A potential consideration would be to separate the architecture of the system into multiple sub-systems that would have different security authorizations or privileges based on the user’s role. |
| 1. Keep It Simple | Keeping the code and design of a system simple can reduce the potential for errors both in the coding and the execution of the program. This can also possibly lead to a more simplified security that would be needed. |
| 1. Default Deny | By default, access should be denied to a system and only granted once conditions are met from the protection and security system. |
| 1. Adhere to the Principle of Least Privilege | A process should run with only the minimum required privilege that is needed. Elevated privileges should only be used if needed to run a process and should be used for as little as possible. Following this would minimize the risk that an attacker would be able to run processes with the elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Functions that are not used, or calls that are made out of context have the potential to pass along data and cause harm to other systems. Sanitizing all data before it is passed to another system can catch the harmful data, such as a potential SQL injection, before the other system uses the data. |
| 1. Practice Defense in Depth | Understanding that one layer of security may not cover all potential vulnerabilities, and that multiple layers are needed to minimize the risk of an attack being successful should one of the layers fail. |
| 1. Use Effective Quality Assurance Techniques | Using proper testing techniques, like penetration testing and fuzzing, as well as performing audits on the code should be implemented as part of an effective quality assurance program. Performing internal and external security reviews can also help in identifying potential issues that would need to be corrected. |
| 1. Adopt a Secure Coding Standard | Making sure coding standards for the language and platform being used are being followed will assist in being secure and reduces the risk of vulnerabilities. |

### 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] | Follow the one-definition rule |

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

| **Noncompliant Code** |
| --- |
| There are two different translation units that define a class named S, but are defined using different sequence of tokens. |
| // a.cpp  struct S {  int a;  };  //b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| Use a header file to make the same class definition visible within both translation units. |
| // S.h  struct S {  int a;  };  // a.cpp  #include “S.h”  // b.cpp  #include “S.h” |

| **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** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | LANG.STRUCT.DEF.FDH  LANG.STRUCT.DEF.ODH | Function defined in header file  Object defined in header file |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Fully implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| Polyspace Bug Finder | R2023a | CERT C++: DCL60-CPP | Checks for inline constraints not respected (rule partially covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Only read initialized memory |

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

| **Noncompliant Code** |
| --- |
| An uninitialized local variable is being used as part of an expression to print its value, which will result in undefined behavior. |
| #include <iostream>  void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The local variable i is initialized before it is used as part of the expression. |
| #include <iostream>  void f() {  int i = 0;  std::cout << i;  } |

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | uninitialized-read | Partially checked |
| Helix QAC | 2023.3 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 |  |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Polyspace Bug Finder | R2023a | CERT C++: EXP53-CPP | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | You should not try 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>

| **Noncompliant Code** |
| --- |
| std::getenv() returns a null pointer, so when std::string object is created from a call to this pointer it results in undefined behavior |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv(“TMP”));  if(!tmp.empty()) {  //...  }  } |

| **Compliant Code** |
| --- |
| First the call to std::getenv() is checked to see if it is null before being used in creating the std::string object. |
| #include <cstdlib>  #include <string>  void f() {  const char \*tmpPtrVal = std::getenv(“TMP”);  std::string tmp(tmpPtrVal ? tmpPtrVal : “”);  if(!tmp.empty()) {  //...  }  } |

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | assert\_failure |  |
| Helix QAC | 2023.3 | DF4770, DF4771, DF4772, DF4773, DF4774 |  |
| Klocwork | 2023.3 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.CALL.MUST  NPD.CHECK.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.CONST.DEREF  NPD.FUNC.CALL.MIGHT  NPD.FUNC.CALL.MUST  NPD.FUNC.MIGHT  NPD.FUNC.MUST  NPD.GEN.CALL.MIGHT  NPD.GEN.CALL.MUST  NPD.GEN.MIGHT  NPD.GEN.MUST  RNPD.CALL  RNPD.DEREF |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

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

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

| **Noncompliant Code** |
| --- |
| Data inputted from a user could maliciously change the query if precautions are not taken. |
| uName = getRequestString(“username”);  uPass = getRequestString(“userpassword”);  sql = “SELECT \* FROM Users WHERE Name = “ + uName + “ AND Pass = “ +  uPass + “ |

| **Compliant Code** |
| --- |
| By using sanitization and validation on inputs, SQL injections can be prevented. This is typically done with stored procedures and parameterized queries. |
| PreparedStatement pStmt = PreparedStatement();  std::cin >>username;  std::cin userpassword;  sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;” {username,  userpassword}; |

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| The Checker Framework | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| Fortify | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented |
| Parasoft Jtest | 2023.1 | **CERT.IDS00.TDSQL** | Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Freed memory should not be accessed. |

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

| **Noncompliant Code** |
| --- |
| After it is deallocated, s is dereferenced and if accessing it results in a write-after-free, arbitrary code can be exploited and ran with permissions of the vulnerable process. |
| #include <new>  struct S {  void f();  };  void g() noexcept(false) {  S \*s = new S;  //...  delete s;  //...  s ->f();  } |

| **Compliant Code** |
| --- |
| Deallocation is not done until the dynamically allocated memory is no longer required. |
| #include <new>  struct S {  void f();  };  void g() noexcept(false) {  S \*s = new S;  //...  s->f();  delete s;  } |

| **Principles(s):**  2: Heed Compiler Warnings  5: Default Deny  6: Adhere to the Principle of Least Privilege  9: Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 7.4p0 | **ALLOC.UAF** | Use after free |
| Coverity | v7.5.0 | **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 C/C++test | 2023.1 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |

#### 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>

| **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(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error “Structure must not have any padding”  #endif |

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.4p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| ECLAIR | 1.2 | **CC2.DCL03** | Fully implemented |
| LDRA tool suite | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

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

Reference: <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  }  // ...  } |

| **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 | **P4** | **L3** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **stdlib-use** | Partially checked |
| CodeSonar | 7.4p0 | **BADFUNC.ABORT** **BADFUNC.EXIT** | Use of abort Use of exit |
| LDRA tool suite | 9.7.1 | **122 S** | Enhanced Enforcement |
| Polyspace Bug Finder | R2023a | CERT C++: ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Object Oriented**  **Programming** | [STD-008-CPP] | Write constructor member initializers in the canonical order |

Reference: <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) {}  }; |

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **initializer-list-order** | Fully checked |
| CodeSonar | 7.4p0 | **LANG.STRUCT.INIT.OOMI** | Out of Order Member Initializers |
| LDRA tool suite | 9.7.1 | **206 S** | Fully implemented |
| RuleChecker | 22.10 | **initializer-list-order** | Fully checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Containers** | [STD-009-CPP] | Use valid iterator ranges |

Reference: <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; });  } |

| **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 | **P6** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **overflow\_upon\_dereference** |  |
| CodeSonar | 7.4p0 | **LANG.MEM.BO** | Buffer Overrun |
| 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 | R2023a | CERT C++: CTR53-CPP | Checks for invalid iterator range (rule partially covered). |

#### Coding Standard 10

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

Reference: <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;  } |

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **return-reference-local** **dangling\_pointer\_use** | Partially checked |
| Clang | 3.9 | **-Wdangling-initializer-list** | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |
| CodeSonar | 7.4p0 | **IO.UAC** **ALLOC.UAF** | Use after close Use after free |
| LDRA tool suite | 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.

When you take action to include security during each step in the DevOps toolchain, you transform it into DevSecOps. During the Assess and Plan step, you would implement threat modeling. During the steps of Design and Build, you would address IDE security. Last in the Pre-production portion during Verify and test, Using static testing and running security scans along with unit and integration testing would be implemented.

When transitioning to Production, use of integrity checks and implementing measures such as defense-in-depth are used. Also, Implementing continuous threat detection methods like penetration testing and network monitoring will assist in security.

### 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 | Medium | Unlikely | Medium | Low (4) | 3 |
| STD-009-CPP | High | Probable | High | Medium (6) | **2** |
| 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 in rest | This protects data that is stored. Some examples of where this may be seen is within hard drives, on phones, cloud assets, and others. To be able to protect this data, the use of encryption tools and security for physical devices is recommended. |
| Encryption at flight | This protects the data that is moving. This may include data that is moving between devices on the same network or could even be data that is being sent out of the network. The data in this case can be protected using measures like email encryption and strong network security that includes the use of protections like firewalls and user authentication. |
| Encryption in use | This is protecting the data that is currently accessed, such as files currently open, databases, or RAM data. Implementing security such as Single-Sign On and Multi Factor Authentication can help protect this type of data. |

Source: [https://cyscale.com/blog/types-of-encryption/](https://cyscale.com/blog/types-of-encryption/%20e,%20at%20Rest%20Data%20-%20Cyscale)

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is verifying that a user is who they are claiming to be. This may look like using username/password security to access systems, using Single-Sign On, and possibly even biometrics like facial recognition to verify someone’s identity. |
| Authorization | While authentication is confirming the identity of a user, authorization is managing the access rights and different privileges. Having this in place can help minimize the possibility of someone accessing data that may be intended to only be accessed by certain people or could limit what can be done to that data based on permissions. |
| Accounting | The process of accounting is making sure that all activity while a system is in use is being monitored in some way. This could include logs that show what was accessed, when it was accessed, who accessed it, and what was done with the data. This is important as it can help verify that both Authentication and Authorization is being implemented properly, and if there happened to be a breach, could potentially assist in seeing the scope of the breach. |

**\***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 | 09/17/2023 | Project 1 milestone | Thomas Bartlett |  |
| 1.2 | 10/08/2023 | Project 1 | Thomas Bartlett |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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