**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 | When handling input data, it is important to check that it is valid, meaning that it has the right format and data type, and doesn’t contain any links, console commands, or malicious code fragments. This can be accomplished by rejecting any input that isn’t whitelisted or that the program isn’t designed to handle. |
| 1. Heed Compiler Warnings | Compiler warnings are given when something is wrong with the software program to be compiled. If part of the code is missing, broken, or using the wrong format, it needs to be fixed before the program can be considered complete. |
| 1. Architect and Design for Security Policies | Security Architecture and Design involves proactively including security in the initial construction of a software program. A program should be designed and built with security in mind, so that it is easier to secure it against malicious attacks. |
| 1. Keep It Simple | Don’t overcomplicate the code you use. Complicated code is more likely to contain flaws and exploits, because it’s hard to understand even for the person who built it. Make sure the code you use can be easily understood, so that other programmers can more easily check it for errors. |
| 1. Default Deny | If part of a system isn’t meant to be accessible to everyone, deny everyone who doesn’t have the authorization to access it. Use a whitelist to only allow people who are specifically allowed, rather than blacklisting the people who specifically aren’t allowed. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege states that users of a system should only have access to the data and resources they absolutely need to perform their designated tasks. If a user doesn’t need a particular resource, they shouldn’t have access. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data involves cleaning or modifying data to remove potentially harmful elements or characters which could cause vulnerabilities or errors. This should be done before sending data to other systems so that malicious attacks aren’t propagated onwards. |
| 1. Practice Defense in Depth | Defense in depth involves using multiple redundant layers of security which different strengths and weaknesses in order to cover as many different potential vulnerabilities as possible. This means that as many different methods of cybersecurity as possible must be used in order to give the best possible protection against cybersecurity attacks. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance involves testing a software program or other product in order to make sure it completes all the requirements and standards it was designed to, with no flaws or errors that would prevent it from functioning properly. |
| 1. Adopt a Secure Coding Standard | A secure coding standard is a set of rules and guidelines that help software developers write code that is resistant to software vulnerabilities. They promote consistent security practices and reduce the number of errors that could potentially compromise the software security of the program. |

### 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-STR50-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

| **Noncompliant Code** |
| --- |
| The input is unbounded and the code could lead to a buffer overflow. |
| #include <iostream>   void f() {  char buf[12];  std::cin >> buf; } |

| **Compliant Code** |
| --- |
| Use std::string instead of a bounded array. |
| #include <iostream> #include <string>   void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo; } |

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

| **Principles(s):** Principle 1, Validate Input Data, maps to this standard because it focuses on making sure the data being entered into the program’s storage doesn’t cause issues by having invalid or incompatible formatting. Strings with variable lengths are generally because of user input, so this principle is the best fit. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| CodeSonar | 9.0p0 | **MISC.MEM.NTERM** **LANG.MEM.BO** **LANG.MEM.TO** | No space for null terminator Buffer overrun Type overrun |
| LDRA tool suite | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-DCL60-CPP] | Obey the one-definition rule. |

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

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

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

| **Principles(s):** Principle 4, Keep it Simple, maps to this standard because it focuses on preventing multiple overlapping definitions from being used in a single program, which could confuse both the programmer and the compiler in different ways if the definitions aren’t the same. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **type-compatibility** **definition-duplicate** **undefined-extern** **undefined-extern-pure-virtual** **external-file-spreading** **type-file-spreading** | Partially checked |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-DCL60-a** | The One Definition Rule shall not be violated |
| CodeSonar | 9.0p0 | **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 |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-STR53-CPP] | Range check element access. |

| **Noncompliant Code** |
| --- |
| The value returned by the call to get\_index is greater than the number of elements stored in the string. |
| #include <string>   extern std::size\_t get\_index();   void f() {  std::string s("01234567");  s[get\_index()] = '1'; } |

| **Compliant Code** |
| --- |
| Use the std::basic\_string::at() function, which throws an exception instead of causing undefined behavior. |
| #include <stdexcept> #include <string> extern std::size\_t get\_index();  void f() {  std::string s("01234567");  try {  s.at(get\_index()) = '1';  } catch (std::out\_of\_range &) {  // Handle error  } } |

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

| **Principles(s):** Principle 3, Architect and Design for Security Policies, maps to this standard because it involves implementing checks to make sure the code is running properly and avoiding a potential path to causing undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **assert\_failure** | N/A |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| CodeSonar | 9.0p0 | **LANG.MEM.BO** **LANG.MEM.BU** **LANG.MEM.TBA** **LANG.MEM.TO** **LANG.MEM.TU** | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |
| Polyspace Bug Finder | R2024b | CERT C++: STR53-CPP | Checks for:  -Array access out of bounds  -Array access with tainted index  -Pointer dereference with tainted offset  Rule partially covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-FIO30-C] | Exclude user input from format strings. |

| **Noncompliant Code** |
| --- |
| A function accepts and uses untrusted data that comes from an unauthenticated user. |
| #include <stdio.h> #include <stdlib.h> #include <string.h>   void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {   /\* Handle error \*/   } else if (ret >= len) {   /\* Handle truncated output \*/   }  fprintf(stderr, msg);  free(msg); } |

| **Compliant Code** |
| --- |
| Outputting the data directly to stderr without evaluating it fixes the problem. |
| #include <stdio.h> #include <stdlib.h> #include <string.h>   void incorrect\_password(const char \*user) {  int ret;  /\* User names are restricted to 256 or fewer characters \*/  static const char msg\_format[] = "%s cannot be authenticated.\n";  size\_t len = strlen(user) + sizeof(msg\_format);  char \*msg = (char \*)malloc(len);  if (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  if (ret < 0) {   /\* Handle error \*/   } else if (ret >= len) {   /\* Handle truncated output \*/   }  fputs(msg, stderr);  free(msg); } |

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

| **Principles(s):** Principle 1, Validate Input Data, maps to this standard very explicitly, with both the principle and standard focusing on the process of validating user data and making sure there are no issues with its format or content. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Very high (Not repairable) | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 |  | Supported via stubbing/taint analysis |
| Parasoft C/C++test | 2024.2 | **CERT\_C-FIO30-a** **CERT\_C-FIO30-b** **CERT\_C-FIO30-c** | Avoid calling functions printf/wprintf with only one argument other than string constant Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable Never use unfiltered data from an untrusted user as the format parameter |
| CodeSonar | 9.0p0 | **IO.INJ.FMT** **MISC.FMT** | Format string injection Format string |
| LDRA tool suite | 9.7.1 | **86 D** | Partially Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-MEM50-CPP] | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| A pointer is dereferenced after it is deallocated. |
| #include <new>   struct S {  void f(); };   void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f(); } |

| **Compliant Code** |
| --- |
| The dynamically allocated 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):** Principle 10, Adopt a Secure Coding Standard, maps to this standard because it focuses on making sure memory pointers aren’t used in ways that they are not intended for. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **dangling\_pointer\_use** | N/A |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |
| CodeSonar | 9.0p0 | **ALLOC.UAF** | Use after free |
| LDRA tool suite | 9.7.1 | **483 S, 484 S** | Partially implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-ERR50-CPP] | Do not abruptly terminate the program. |

| **Noncompliant Code** |
| --- |
| A function call may result in a call to std::terminate() |
| #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 function handles all exceptions thrown and doesn’t 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):** Principle 2, Heed Compiler Warnings, maps to this standard because it makes sure the program actually handles the exceptions instead of ignoring them and cutting the program short. |
| --- |

**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 |
| Parasoft C/C++test | 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 |
| CodeSonar | 9.0p0 | **BADFUNC.ABORT** **BADFUNC.EXIT** | Use of abort Use of exit |
| LDRA tool suite | 9.7.1 | **122 S** | Enhanced Enforcement |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-ERR51-CPP] | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| Thrown exceptions are not caught in the code. |
| void throwing\_func() noexcept(false);   void f() {  throwing\_func(); }   int main() {  f(); } |

| **Compliant Code** |
| --- |
| The main entry point handles all exceptions. |
| void throwing\_func() noexcept(false);   void f() {  throwing\_func(); }   int main() {  try {  f();  } catch (...) {  // Handle error  } } |

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

| **Principles(s):** Principle 2, Heed Compiler Warnings, maps very explicitly to this standard since they are both focused on handling the exceptions the programs encounters and outputs to the user. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions 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 |
| CodeSonar | 9.0p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| LDRA tool suite | 9.7.1 | **527 S** | Partially implemented |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | [STD-ERR55-CPP] | Honor exception specifications. |

| **Noncompliant Code** |
| --- |
| A function is declared as non-throwing, but it is possible for it to throw an exception when the requested memory can’t be allocated. |
| #include <cstddef> #include <vector>   void f(std::vector<int> &v, size\_t s) noexcept(true) {  v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| The noexcept-specification is removed, making the function allow all exceptions. |
| #include <cstddef> #include <vector>  void f(std::vector<int> &v, size\_t s) {  v.resize(s); // May throw, but that is okay } |

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

| **Principles(s):** Principle 10, Adopt a Secure Coding Standard, maps to this standard because it is focused on making sure a part of the code isn’t used in a way that will cause problems with the program’s execution. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **unhandled-throw-noexcept** | Partially checked |
| Parasoft C/C++Test | 2024.2 | **CERT\_CPP-ERR55-a** | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |
| CodeSonar | 9.0p0 | **LANG.STRUCT.EXCP.THROW** | Use of throw |
| Polyspace Bug Finder | R2024b | CERT C++: ERR55-CPP | Checks for noexcept functions exiting with exception (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Protection | [STD-EXP53-CPP] | Do not read uninitialized memory. |

| **Noncompliant Code** |
| --- |
| An uninitialized local variable is evaluated. |
| #include <iostream>   void f() {  int i;  std::cout << i; } |

| **Compliant Code** |
| --- |
| The object is initialized before being evaluated. |
| #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):** Principle 10, Adopt a Secure Coding Standard, maps to this standard because it is focused on making sure a part of the code isn’t used in a way that will cause the program to fail or attempt to evaluate an undefined object. |
| --- |

**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 |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-EXP53-a** | Avoid use before initialization |
| CodeSonar | 9.0p0 | **LANG.STRUCT.RPL** **LANG.MEM.UVAR** | Return pointer to local Uninitialized variable |
| LDRA tool suite | 9.7.1 | **53 D, 69 D, 631 S, 652 S** | Partially implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Type | [STD-EXP50-CPP] | Do not depend on the order of evaluation for side effects. |

| **Noncompliant Code** |
| --- |
| An integer is evaluated more than once in an unsequenced manner. |
| void f(int i, const int \*b) {  int a = i + b[++i];  // ... } |

| **Compliant Code** |
| --- |
| The operations act independently of the order of operations and can only be interpreted one way. |
| void f(int i, const int \*b) {  ++i;  int a = i + b[i];  // ... } |

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

| **Principles(s):** Principle 10, Adopt a Secure Coding Standard, maps to this standard because most secure coding standards make sure that mathematical operations in code are written out explicitly with parentheses to clarify where necessary, which is the best way to avoid encountering the type of error this standard is meant to prevent. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-EXP50-a** **CERT\_CPP-EXP50-b** **CERT\_CPP-EXP50-c** **CERT\_CPP-EXP50-d** **CERT\_CPP-EXP50-e** **CERT\_CPP-EXP50-f** | The value of an expression shall be the same under any order of evaluation that the standard permits Don't write code that depends on the order of evaluation of function arguments Don't write code that depends on the order of evaluation of function designator and function arguments Don't write code that depends on the order of evaluation of expression that involves a function call Between sequence points an object shall have its stored value modified at most once by the evaluation of an expression Don't write code that depends on the order of evaluation of function calls |
| LDRA tool suite | 9.7.1 | **35 D, 1 Q, 9 S, 134 S, 67 D, 72 D** | Partially implemented |
| ECLAIR | 1.2 | **CC2.EXP30** | Fully implemented |
| Polyspace Bug Finder | R2024b | CERT C++: EXP50-CPP | Checks for situations where expression value depends on order of evaluation (rule fully covered). |

### 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.

To automate enforcement of the security standards in this policy, the existing DevOps process should be modified to include security in each step; rather than just adding steps to the process which focus on security, the focus of the existing steps should be shifted towards building a secure application. There is one part of the development process that should be added to test and assess the security of the application so that it can be improved further in the following steps, which should be added after the build phase. The production phase of application development also requires additional steps to make sure the secure application stays secure once it has been deployed, which results in another development cycle being linked to the first. A major part of DevSecOps is to design the application to be secured against malicious attacks, but the other major difference compared to DevOps is that the application still needs to be improved security-wise even after it has been completed and released to the public.

### 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-STR50-CPP] | High | Likely | Medium | P18 | L1 |
| [STD-DCL60-CPP] | High | Unlikely | High | P3 | L3 |
| [STD-STR53-CPP] | High | Unlikely | Medium | P6 | L2 |
| [STD-FIO30-C] | High | Likely | Very high (Not repairable) | P18 | L1 |
| [STD-MEM50-CPP] | High | Likely | Medium | P18 | L1 |
| [STD-ERR50-CPP] | Low | Probable | Medium | P4 | L3 |
| [STD-ERR51-CPP] | Low | Probable | Medium | P4 | L3 |
| [STD-ERR55-CPP] | Low | Likely | Low | P9 | L2 |
| [STD-EXP53-CPP] | High | Probable | Medium | P12 | L1 |
| [STD-EXP50-CPP] | Medium | Probable | Medium | P8 | L2 |

### 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 involves encrypting data that is stored somewhere, such as on a hard drive or SSD. This type of encryption should be used to protect the stored data against unauthorized access in case the storage media is stolen or compromised, and involves encrypting the stored data in some way using an encryption key that must also be protected. |
| Encryption in flight | This involves encrypting data that is being transmitted over networks, such as the internet or private networks. This type of encryption is meant to prevent the data from being intercepted while in transit and stop eavesdropping or man-in-the-middle attacks, and uses symmetric encryption algorithms and secure communication protocols. |
| Encryption in use | This involves protecting data that is actively being processed by applications, such as in processing units. This type of encryption is used to prevent the unauthorized access or modification of data while it is being used, and is the most difficult type of encryption, since some of the data the application may try to access should be in use, and so cannot remain encrypted all the time. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the initial step of the security framework, where the identity of the user or device attempting to access the network is verified. This step is designed to make sure the entity is actually who they claim to be, and is usually accomplished with passwords, multifactor authentication, and digital certificates. |
| Authorization | Authorization comes after successful authentication, and is used to determine what the authenticated entity is allowed to do within the network. This step involves granting or denying access to resources and defining what parts of the network a user is able to access, and generally uses either role-based or policy-based access control. |
| Accounting | Accounting is the last of these three security policies, which focuses on tracking and recording the user’s activities and resource usage. This step logs quite a bit of important information to be used later in security audits, compliance checks, detecting anomalies, and billing. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 05/26/2025 | 3-2 Milestone | Michael Oelschlager | Prof. Kraya |
| 3.0 | 06/14/2025 | 6-2 Project One | Michael Oelschlager | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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