**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 | Validating input data is essential in ensuring that information being taken into a program from users is correct, will not cause any unexpected behavior such as buffer underflows or overflows which can compromise the integrity of the program and possibly allow unauthorized access to the system to include the retrieval of sensitive data. |
| 1. Heed Compiler Warnings | Compiler warnings are meant to inform the developers when there is a potential issue that can cascade into multiple failures, causing unexpected behavior or data breaches. Proper investigation into any warnings that appear is essential to catching and resolving issues early on in development so that the issues do not occur when an application is live. |
| 1. Architect and Design for Security Policies | Security should be at the forefront of any software design, not an afterthought. Security features should be built into the design at every level to mitigate the risk of attacks. Code like an attack will happen, not like it might happen. |
| 1. Keep It Simple | Errors in overly complicated code are more difficult to find and fix and roundabout ways of accomplishing a task waste valuable memory resources. Keep the path from initialization to termination as straightforward as possible and utilize standard libraries to reduce the number of functions that are hand coded, limiting risk of developer error even more. |
| 1. Default Deny | The only way a user should be able to access a system is through proper authentications. If they are not authenticated, they should be denied. This goes for any user-driven functions as well, if they do not have permissions, it should be an automatic denial with no way around it. |
| 1. Adhere to the Principle of Least Privilege | Users should only have the bare minimum permissions required to perform the task they are meant to. A customer does not need the ability to delete a product from the database. This ensures nobody has access beyond what they are trusted and required to do. |
| 1. Sanitize Data Sent to Other Systems | Any time data is output to a client device or another server, it must be checked to ensure that there is no sensitive data that is being transferred that should not be. Ensuring that there is no code that leaks data to unauthorized users is paramount in maintaining integrity and the security of the company’s and customers’ data. |
| 1. Practice Defense in Depth | No single method or tool can protect against every type of attack. Defense in Depth is meant to maximize coverage and protection for the system and the data contained within. |
| 1. Use Effective Quality Assurance Techniques | Test as extensively as possible and manually review code to check for errors that could lead to failure. Continue to periodically audit code after launch to ensure that the code continues to meet the standard for security. |
| 1. Adopt a Secure Coding Standard | People’s definitions of what is “secure” may differ. Create guidelines that everyone must follow to make sure that the company’s standard for security is upheld. This also assists in the event that the original developer leaves the company, anyone else can pick up where they left off. |

### 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] | Validate user input for proper data types. |

| **Noncompliant Code** |
| --- |
| Does not validate that the input matches the assigned data type for a variable. |
| Int result;  Int userInput;  Std::cin >> userInput;  Result = 5 + userInput |

| **Compliant Code** |
| --- |
| Validates that user input matches the assigned data type for the variable or does not allow the input. |
| Int result;  Int userInput;  while (!(std::cin >> userInput)) {  /\* clear buffer and take new input or quit \*/  }  Result = 5 + userInput; |

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

| **Principles(s):** Validate Input Data. This is an elementary validation that needs to be done to prevent programs crashing. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Stdlib-use | Partially checked |
| CodeSonar | 8.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort Use of exit |
| 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 |
| Polyspace Bug Finder | R2024a | CERT C++: ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use data types that dynamically allocate memory. |

| **Noncompliant Code** |
| --- |
| Using character arrays for string data. |
| Char userPass[20];  Std::cin >> userPass; |

| **Compliant Code** |
| --- |
| Instead use std::string for character input and validate length of string after for length sensitive data |
| Std::string userPass;  Std::cin >> userPass;  while (userPass.length() >= (maxPassSize + 1)) {  /\* clear buffer and take new input \*/ }  /\*continue authentication\*/ |

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

| **Principles(s):** Keep it simple. Excessive need to program for memory management and protecting against buffer overflows will get complicated. Let the system do what it is designed to do and allocate the memory required, self managing memory. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 4.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| LDRA tool suite | 9.7.1 | F5 D | Partially implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| Polyspace Bug Finder | R2024a | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Format Strings with user input when entering into pre-written sentences. |

| **Noncompliant Code** |
| --- |
| Taking input directly into the use variable. |
| Std::string name;  Int age;  Std::cout << name << “ is “ << age << “ years old.” << std::endl; |

| **Compliant Code** |
| --- |
| Use std::format which checks types and provides a level of protection against buffer overflow attacks. |
| Std::string name;  Int age;  Std::string newString = std::format(“{} is {} years old”, name, age);  Std::cout << newString << endl; |

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

| **Principles(s):** Sanitize Data Sent to Other Systems. This goes for any client-server transfer, client to client transfer, or anytime a user inputs information into the system. Ensuring that data transfer will not compromise our network. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | 2023.1 | CERT C: Rule STR30-C | A string literal shall not be modified  Do not modify string literals |
| RuleChecker | 24.04 | string-literal-modfication | Partially Checked |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Exhaustively verified |
| PC-lint Plus | 1.4 | 489, 1776 | Partially supported |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Check for SQL code in user input and reject if found. |

| **Noncompliant Code** |
| --- |
| Does not validate user input. |
| Std::string userPass;  Std::cin >> userpass;  char\* error\_message;  if (sqlite3\_exec(db, userPass.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true; |

| **Compliant Code** |
| --- |
| Uses std::rejex to run pattern recognition to check for SQL code in input |
| Std::string userPass;  Std::cin >> userPass;  std::regex injection\_pattern(R"((\b(OR|AND)\b\s\*(\d+\s\*=\s\*\d+|'[^']\*'\s\*=\s\*'[^']\*'))|\b(UNION|SELECT|INSERT|UPDATE|DELETE)\b|(--)|(;))", std::regex\_constants::icase);  if (std::regex\_search(sql, injection\_pattern)) {  /\* Handle possible attack by rejecting string and returning\*/  }  char\* error\_message;  if (sqlite3\_exec(db, userPass.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true; |

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

| **Principles(s):** Sanitize Data Sent to Other Systems, Default Deny. Any questionable data should be flagged and the user denied access to the system at minimum until the input can be reviewed by a specialist. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | TAINTED\_STRING | Implemented |
| CodeSonar | 8.1p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| CppCheck Premium | 24.9.0 | Premium-cert-fio30-c | Partially implemented |
| Polyspace Bug Finder | R2024a | CERT C: RULE FIO30-C | Checks for tainted string format (rule partially covered) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Clear std::cin buffers when rejecting input. |

| **Noncompliant Code** |
| --- |
| Continuously adds new input into limited space. |
| Char userInput[20]  Bool validInput = false;  Std::cin >> userInput;  While (!validInput) {  Std::cout << “Invalid input, please retry << std::endl;  Std::cin >> userInput;  } |

| **Compliant Code** |
| --- |
| Makes sure that the inputstream buffer is cleared for each attempt. |
| char userInput[20];  bool validInput = false;  std::cin >> userInput;  while (!validInput || userInput != ‘q’) {  std::cout << “Invalid input, please retry or ‘q’ to quit” << endl;  std::cin.clear();  stdLLcin.ignore(100, ‘/n’);  std::cin >> userInput;  } |

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

| **Principles(s):** Architect and Design for Security Policies. Clearing out the cin buffer after it is done is a simple, basic, and standard practice and should be done wherever. This goes beyond the company policy and is just good practice. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Code Sonar | 8.1p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Free non-heap variable  Double free  Type mismatch  Leak |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule. |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Use the same form in corresponding calls to new/malloc and delete/free  Always provide empty brackets ([]) for delete when deallocating arrays  Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor  Properly deallocate dynamically allocated resources |
| Polyspace Bug Finder | R2024a | CERT C++:MEM51-CPP | Checks for:   * Invalid deletion of pointer * Invalid free of pointer * Deallocation of previously deallocated pointer   Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use assertions to assist with validating data during development, best not to use assertions post release. |

| **Noncompliant Code** |
| --- |
| Does not use assertions to cause failures if int is null. |
| Int thisNum;  bool validInput = false;  std::cin >> thisNum;  while (!validInput || thisNum != 0) {  std::cout << “Invalid input, please retry or ‘0’ to quit” << std::endl;  cin.clear();  cin.ignore(100, ‘/n’);  cin >> thisNum;  } |

| **Compliant Code** |
| --- |
| Uses assertions to validate input |
| Int thisNum;  bool validInput = false;  std::cin >> thisNum;  assert (thisNum != NULL);  while (!validInput || thisNum != 0) {  std::cout << “Invalid input, please retry or ‘0’ to quit” << std::endl;  cin.clear();  cin.ignore(100, ‘/n’);  cin >> thisNum;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques. Utilizing assertions is good for testing data and can be done to verify expected output and behavior and are effective testing tools. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/Rose |  |  | Can detect some violations of this rule. However it can only detect violations involving abort() because assert() is implemented as a macro |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced enforcement |
| Parasoft C/C++ test | 2023.1 | CERT\_C-ERR06-a | Do not use assertions |
| PC-lint Plus | 1.4 | 586 | Fully supported |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| If exception is thrown, aborts program. |
| Int start = 0;  Int increment = 3;  Int result = start;  Try {  Result = start + increment;  Std::cout << “adding numbers” << std::endl;  while (1) {  result = result + increment;  if (result > 10) {  throw std::exception(“Error”);  }  }  Catch (std::exception& excpt) {  Abort();  } |

| **Compliant Code** |
| --- |
| Handle exceptions gracefully and appropriately; |
| Int start = 0;  Int increment = 3;  Int result = start;  Int maxNum;  Bool lessThan  Try {  Result = start + increment;  Std::cout << “adding numbers” << std::endl;  while (lessThan) {  result = result + increment;  if (result > maxNum) {  throw std::exception(“Error”);  }  }  Catch (std::exception& excpt) {  Std::cout << “Addition surpassed limit” << std::endl;  Return;  } |

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

| **Principles(s):** Architect and Design for Security Policies. This is the developer’s opportunity to anticipate and catch errors before they occur so that the system can handle them and prevent the system from failing and program crashing due to a foreseeable event. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Polyspace Bug Finder | R2024a | CERTC++:ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered) |
| RuleChecker | 22.10 | Stdlib-use | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programing | [STD-008-CPP] | Write constructor member initializers in declaration order. |

| **Noncompliant Code** |
| --- |
| Initializes values in reverse order. |
| Class thisClass {  Int value1;  String value2;  Bool value3;  Public thisClass {  Value3 = false;  Value2 = “Hello”;  Value1 = 67;  }  } |

| **Compliant Code** |
| --- |
| Constructor initializes values in declarative order. |
| Class thisClass {  Int value1;  String value2;  Bool value3;  Public thisClass {  Value1 = 67;  Value2 = “Hello”;  Value3 = false;  }  } |

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

| **Principles(s):** Architect and Design for Security Policies, Keep it simple. While seemingly a small thing, keeping constructors in declarative order can minimize mistakes such as not initializing a variable. It also improves readability for later maintenance and updates on the system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Initializer-list-order | Fully checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.INIT.OOMI | Out of Order Member Initializers |
| LDRA tool suite | 9.7.1 | 206 S | Fully implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-OOPS3-A | List members in an initialization list in the order in which they are declared |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-009-CPP] | Use valid ranges for iterating through containers. |

| **Noncompliant Code** |
| --- |
| Will likely iterate incorrect number of times, either too many times or too little. |
| Int numElements;  Std::cin >> numElements;  Vector<int> thisVector(numElements);  For (int i = 0; i < 100; ++i( {  Std::cout << thisVector.at(i);  } |

| **Compliant Code** |
| --- |
| Iterates for the number of elements in the vector. |
| Int numElements;  Std::cin >> numElements;  Vector<int> thisVector(numElements);  For (int i = 0; i < thisVector.size(); ++i( {  Std::cout << thisVector.at(i);  } |

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

| **Principles(s):** Heed Compiler Warnings, Architect and Design for Security Policies, Default Deny, Practice Defense in Depth. Letting an iterator go free will absolutely cause errors. Limiting iterators to their range will protect the system from crashing down on itself because of a function trying to pull from data that it is not supposed to or overwriting existing memory. Compilers should warn about this error, make sure to pay attention. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.MEM.BO | Buffer Overrun |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP.CTR53-a  CERT\_COO-CTR53-b | Do not use an iterator range that isn’t really a range.  Do not compare iterators from different containers. |
| Polyspace Bug Finder | R2024a | CERT C++: CTR53-CPP | Checks for invalid iterator range (rule partially covered) |
| Helix QAC | 2024.2 | C++3802 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-010-CPP] | Do not access an object before it is initialized or after it is destroyed. |

| **Noncompliant Code** |
| --- |
| Accesses object before initialization. |
| Class thisClass {  private Int classId = 1;  Public Int getClassId() {  Return this.classId;  }  }  Int main {  Int myNum;  compareNum = thisClass.getClassId();  Std::cin >> myNum;  newClass = new thisClass;  if (myNum > compareNum) {  std::cout << “I win” << std::endl;  }  } |

| **Compliant Code** |
| --- |
| Creates object before accessing object class member. |
| Class thisClass {  private Int classId = 1;  Public Int getClassId() {  Return this.classId;  }  }  Int main {  newClass = new thisClass;  Int myNum;  compareNum = thisClass.getClassId();  Std::cin >> myNum;  if (myNum > compareNum) {  std::cout << “I win” << std::endl;  }  } |

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

| **Principles(s):** Heed Compiler Warnings, Use Effective Quality Assurance Techniques. Attempting to call an object member before the object has been created will cause an error that will cause the system to crash, attempting to call an object member after the object has been destroyed will pull information from somewhere else in the system memory if the system does not catch it and shut down the program first. This can give free access to private methods that could be exploited to gain unauthorized access to confidential data. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Return-reference-local | Partially checked |
| 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 |
| LDRA tool suite | 9.7.1 | 42 D, 53 D, 77 D, 1 J, 71 S, 565 S | Partially implmented |

### 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.

Automations should be implemented within the Build and Verify and Test sections. Adding them to the Requiring developers to utilize automated code checking during the build process will allow vulnerabilities to be caught early on in the build and fixed reducing the remediation costs and testing of individual functionality for adherence to the standards. Including automated checking during the Verify and Test phase will allow the checkers to asses the program as a whole including the connections between files and ensure that the entire program meets the standards set forth.

### 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 | Low | 2 |
| [STD-002-CPP] | High | Likely | Medium | High | 1 |
| [STD-003-CPP] | Low | Likely | Low | Medium | 2 |
| [STD-004-CPP] | High | Likely | Medium | High | 1 |
| [STD-005-CPP] | High | Likely | Medium | High | 1 |
| [STD-006-CPP] | Medium | Unlikely | Medium | Low | 3 |
| [STD-007-CPP] | Low | Probable | Medium | Low | 3 |
| [STD-008-CPP] | Medium | Unlikely | Medium | Low | 3 |
| [STD-009-CPP] | High | Probable | High | Medium | 2 |
| [STD-010-CPP] | High | Probable | High | Medium | 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 | User and company data should be encrypted while being stored on hard drives or databases to mitigate the risk of security breaches compromising confidential data. User passwords should be stored using hashing or by another one-way operation. |
| Encryption in flight | Data that is being transmitted between devices should be encrypted using an RSA encryption algorithm. This will mitigate the risk of an eavesdropper gaining access to confidential information. |
| Encryption in use | Data that is actively being processed should be raw to limit the chances of miscalculations or manipulations. Because of this other safeguards must be put in place to secure the data being processed. The company’s single sign-on and multi-factor authentication are in place in order to protect this data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Each employee will have a unique password and have single sign-on permissions associated with their account. They will also be required to used unique physical security keys with a security pin for multi-factor authentication to be able to access the system to ensure that anyone who is able to access the network, should be accessing the network. |
| Authorization | No user will have any more access than is absolutely necessary to do their job. Temporary exceptions can be made with manager approval for unique circumstances with business justification. Each user will be added to LDAP groups based on their organization tree and role to manage permissions. This especially includes access to confidential files and ability to modify company databases. |
| Accounting | User activity will be logged in a private database for accountability and investigation purposes. This log will not be accessible to anyone who is not a manager and managers will only be able to see the activities of those below them in the organization tree to ensure that private data is kept private. |

**\***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 |  |
| 1.5 | 09/23/2024 | Milestone 1 | Eric Breznen |  |
| 2.0 | 10/13/2024 | Complete Document | Eric Breznen |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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