**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 | We must validate and sanitize any data received from external sources. |
| 1. Heed Compiler Warnings | It’s important to pay attention to the compiler or static analysis tools. Warnings are there for a reason. |
| 1. Architect and Design for Security Policies | Security considerations must be incorporated into the architecture and design of any system. It’s important to identify and define security policies. |
| 1. Keep It Simple | We must avoid unnecessary complexity in the design and implementation, since more complex system are more prone to vulnerabilities. |
| 1. Default Deny | When granting access adopt a “default deny” approach. We should only be granting necessary permissions. |
| 1. Adhere to the Principle of Least Privilege | Grant users, processes, and systems the minimum set of privileges required. |
| 1. Sanitize Data Sent to Other Systems | Data transmissions must be sanitized and validated. |
| 1. Practice Defense in Depth | Implement multiple layers of security, but not too many that becomes unmanageable. This could be firewalls, intrusion detection etc. |
| 1. Use Effective Quality Assurance Techniques | Incorporate through quality assurance, through the SDLC, perform through testing, this includes functional testing, security testing and vulnerabilities. |
| 1. Adopt a Secure Coding Standard | There are many standards such as OWASP, CERT and others. These standards provide best practices for writing secure code and help developers avoid common security pitfalls. |

### 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 | Range Verification |

| **Noncompliant Code** |
| --- |
| This piece of code has an array with 5 elements, but we are trying to access an element at index 10. |
| int main() {  int myArray[5] = {1, 2, 3, 4, 5};  // Trying to access an index beyond the array's range  int outOfRangeIndex = 10;  std::cout << myArray[outOfRangeIndex] << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| In this version we verify if the index is within the range before accessing the element. |
| int main() {  int myArray[5] = {1, 2, 3, 4, 5};  // Trying to access an index beyond the array's range  int outOfRangeIndex = 10;  if (outOfRangeIndex >= 0 && outOfRangeIndex < 5) {  std::cout << myArray[outOfRangeIndex] << std::endl;  }  else {  std::cout << "Invalid index!" << std::endl;  }  return 0;  } |

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

| **Principles(s):** Then ValidateInput Data principle is quite apparent here since we are ensuring the element that is being access is within range before we even access the element thus validating the input. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **cast-integer-to-enum** | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 7.3p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2023.1 | C++ 3013 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | No Args C-style functions |

| **Noncompliant Code** |
| --- |
| This piece of code does not checks for the argumentst passed into the function. |
| void printNumbers(int numCount, ...) {  va\_list args;  va\_start(args, numCount);  for (int i = 0; i < numCount; i++) {  int number = va\_arg(args, int);  printf("%d ", number);  }  va\_end(args);  } |

| **Compliant Code** |
| --- |
| In this piece of code we added a check that the arguments passed are the intended. |
| void printNumbers(int numCount, ...) {  va\_list args;  va\_start(args, numCount);  for (int i = 0; i < numCount; i++) {  int number = va\_arg(args, int);  printf("%d ", number);  }  va\_end(args);  }  int main() {  int numCount = 3;  int arg1 = 1;  int arg2 = 2;  int arg3 = 3;  if (numCount >= 1 && numCount <= 3) {  printNumbers(numCount, arg1, arg2, arg3);  } else {  printf("Invalid number of arguments!\n");  }  return 0;  } |

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

| **Principles(s):** Keep it simple, being able to check the arguments intended makes it easier to spot issues in the codebase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2023.1 | C++ 3033, C++ 3038 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-EXP56-a | Do not call a function with a mismatched language linkage |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Overflow Prevention. |

| **Noncompliant Code** |
| --- |
| Strcpy copies the string without considering overflows. |
| #include <cstring>  void copyString(char\* dest, const char\* src) {  strcpy(dest, src);  } |

| **Compliant Code** |
| --- |
| In the compliant code we ensure that it fits in the buffer size destSize. |
| #include <cstring>  void copyString(char\* dest, const char\* src, size\_t destSize) {  strncpy(dest, src, destSize - 1);  dest[destSize - 1] = '\0';  } |

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

| **Principles(s):** Architect and Design for Security policies, Security considerations must be incorporated into the architecture and design of any system. It’s important to identify and define security policies. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **invalid\_pointer\_dereference** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **BADFUNC.BO.\* LANG.MEM.BO LANG.MEM.TBA** | A collection of warning classes that report uses of library functions prone to internal buffer overflows. Buffer Overrun Tainted Buffer Access |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF3526, DF3527, DF3528, DF3529, DF3530, DF3531, DF3532, DF3533, DF3534** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-CTR52-a** | Do not pass empty container iterators to std algorithms as destinations |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Explicit code. |

| **Noncompliant Code** |
| --- |
| User input is passed directly into the SQL query instead of properly handling it and sanitizing it. |
| void executeQuery(const std::string& query) {  // Execute the query  std::cout << "Executing query: " << query << std::endl;  }  int main() {  std::string userInput = "'; DROP TABLE Users; --";  // Constructing SQL query without proper handling  std::string query = "SELECT \* FROM Users WHERE username = '" + userInput + "';";  executeQuery(query);  return 0;  } |

| **Compliant Code** |
| --- |
| Query is constructed with parameter queries, thus, the user input is provided as a parameter(data) rather than executable code. This prevents unintended SQL statement executions. |
| void executeQuery(const std::string& query) {  std::cout << "Executing query: " << query << std::endl;  }  int main() {  std::string userInput = "'; DROP TABLE Users; --";  // Constructing SQL query with proper handling  std::stringstream queryStream;  queryStream << "SELECT \* FROM Users WHERE username = ?";  std::string query = queryStream.str();  executeQuery(query);  return 0;  } |

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

| **Principles(s):** 7. Sanitize Data Sent to Other Systems, in this standard we ensure that sql queries are sent exclusively as data rather than executable code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.STRUCT.DECL.FNEST** | Nested Function Declaration |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++1109, C++2510** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **CERT.DCL.AMBIGUOUS\_DECL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **296 S** | Partially implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Unsafe Memory Access |

| **Noncompliant Code** |
| --- |
| The function is called with a nullpointer, deferencing the ptr leads to undefined behavior which could cause a crash. |
| void accessMemory(int\* ptr) {  std::cout << "Value at address " << ptr << ": " << \*ptr << std::endl;  }  int main() {  int\* ptr = nullptr;  accessMemory(ptr);  return 0;  } |

| **Compliant Code** |
| --- |
| We added a null ptr check before accessing memory, if the pointer is null, we print null pointer. Doing so, would prevent dereferencing the null pointer, and avoid unexpected behavior. |
| void accessMemory(int\* ptr) {  if (ptr != nullptr) {  std::cout << "Value at address " << ptr << ": " << \*ptr << std::endl;  } else {  std::cout << "Null pointer!" << std::endl;  }  }  int main() {  int\* ptr = nullptr;  accessMemory(ptr);  return 0;  } |

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

| **Principles(s):** Architect and Design for Security policies, Security considerations must be incorporated into the architecture and design of any system. It’s important to identify and define security policies. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **ALLOC.UAF** | Use After Free |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF4746, DF4747, DF4748, DF4749** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR52-a** | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C++: STR52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr52cpp.html) | Checks for use of invalid string iterator (rule partially covered). |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Improper Assertions |

| **Noncompliant Code** |
| --- |
| The divide function is perform without verifying if the ‘b’ is zero. This is a simple mistake that will break the code due to division by zero. |
| int divide(int a, int b) {  return a / b;  }  int main() {  int a = 10;  int b = 0;  int result = divide(a, b);  std::cout << "Result: " << result << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| We added an assertion to ensure ‘b’ is not zero, if it is an assertion error occurs. |
| int divide(int a, int b) {  assert(b != 0 && "Divisor cannot be zero!");  return a / b;  }  int main() {  int a = 10;  int b = 0;  int result = divide(a, b);  std::cout << "Result: " << result << std::endl;  return 0;  } |

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

| **Principles(s):**  Heed Compiler Warnings - It’s important to pay attention to the compiler or static analysis tools. Warnings are there for a reason.  Architect and Design for Security Policies - Security considerations must be incorporated into the architecture and design of any system. It’s important to identify and define security policies.  Keep It Simple - We must avoid unnecessary complexity in the design and implementation, since more complex system are more prone to vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **MISRA.CATCH.ALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Robust Exception Handling |

| **Noncompliant Code** |
| --- |
| Similar to our previous example this would result in a zero division error. |
| int divide(int a, int b) {  return a / b;  }  int main() {  int a = 10;  int b = 0;  int result = divide(a, b);  std::cout << "Result: " << result << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| The divide function now checks if ‘b’ is zero, if that is the case then we throw a runtime error specifying the issue at hand. |
| #include <stdexcept>  int divide(int a, int b) {  if (b == 0) {  throw std::runtime\_error("Divisor cannot be zero!");  }  return a / b;  }  int main() {  int a = 10;  int b = 0;  try {  int result = divide(a, b);  std::cout << "Result: " << result << std::endl;  } catch (const std::runtime\_error& e) {  std::cout << "Exception caught: " << e.what() << std::endl;  }  return 0;  } |

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

| **Principles(s):**  Heed Compiler Warnings - It’s important to pay attention to the compiler or static analysis tools. Warnings are there for a reason.  Keep It Simple - We must avoid unnecessary complexity in the design and implementation, since more complex system are more prone to vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stdlib-use** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++5014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **MISRA.TERMINATE** **CERT.ERR.ABRUPT\_TERM** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **122 S** | Enhanced Enforcement |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-008-CPP | Variable shadowing |

| **Noncompliant Code** |
| --- |
| In this piece of code we can see that x is being shadowed inside the if conditional statement. |
| int main() {  int x = 5;  if (x > 0) {  int x = 10; // Shadowing the outer 'x' variable  std::cout << "Inner x: " << x << std::endl;  }  std::cout << "Outer x: " << x << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| Contrary to shadowing the outer scope ‘x’ variable we used a different name for the inner variable. This way it becomes clear what is the intention of the code. |
| int main() {  int x = 5;  if (x > 0) {  int innerX = 10; // Unique variable name avoids shadowing  std::cout << "Inner x: " << innerX << std::endl;  }  std::cout << "Outer x: " << x << std::endl;  return 0;  } |

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

| **Principles(s):**  Use Effective Quality Assurance Techniques - Incorporate through quality assurance, through the SDLC, perform through testing, this includes functional testing, security testing and vulnerabilities.  Heed Compiler Warnings - It’s important to pay attention to the compiler or static analysis tools. Warnings are there for a reason.  Keep It Simple - We must avoid unnecessary complexity in the design and implementation, since more complex system are more prone to vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **reserved-identifier** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreserved-id-macro -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++5003** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | **MISRA.DEFINE.WRONGNAME** **MISRA.DEFINE.WRONGNAME.UNDERSCORE** **MISRA.UNDEF.WRONGNAME** **MISRA.UNDEF.WRONGNAME.UNDERSCORE** **MISRA.STDLIB.WRONGNAME** **MISRA.STDLIB.WRONGNAME.UNDERSCORE** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-009-CPP | Proper resource management. |

| **Noncompliant Code** |
| --- |
| The file data.txt is opened for reading, however, it is not explicitly close, this could result in resources leak. |
| void readFile(const std::string& filename) {  std::ifstream file(filename);  if (file.is\_open()) {  std::string line;  while (std::getline(file, line)) {  std::cout << line << std::endl;  }  } else {  std::cout << "Failed to open file: " << filename << std::endl;  }  }  int main() {  std::string filename = "data.txt";  readFile(filename);  return 0;  } |

| **Compliant Code** |
| --- |
| This piece of code is largely similar with the exception of file.close() which we use to implicitly close the filed after we are done reading/using it. |
| void readFile(const std::string& filename) {  std::ifstream file(filename);  if (file.is\_open()) {  std::string line;  while (std::getline(file, line)) {  std::cout << line << std::endl;  }  file.close(); // Close the file after reading  } else {  std::cout << "Failed to open file: " << filename << std::endl;  }  }  int main() {  std::string filename = "data.txt";  readFile(filename);  return 0;  } |

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

| **Principles(s):**  Architect and Design for Security Policies - Security considerations must be incorporated into the architecture and design of any system. It’s important to identify and define security policies.  Use Effective Quality Assurance Techniques - Incorporate through quality assurance, through the SDLC, perform through testing, this includes functional testing, security testing and vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF4786, DF4787, DF4788** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **RH.LEAK** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-010-CPP | Single Responsibility Code |

| **Noncompliant Code** |
| --- |
| The User class is responsible for multiple tasks, including user authentication, saving user data to the database, and sending notifications. Thus, this violates the single responsibility principle. |
| class User {  public:  void login(const std::string& username, const std::string& password) {  // Code for user authentication  // Code for saving user data to the database  // Code for sending notifications to the user  }  }; |

| **Compliant Code** |
| --- |
| Responsibilities are separated into distinct classes. |
| class UserAuthenticator {  public:  void login(const std::string& username, const std::string& password) {  // Code for user authentication  }  };  class UserDatabase {  public:  void saveUserToDatabase(const User& user) {  // Code for saving user data to the database  }  };  class UserNotifier {  public:  void sendNotification(const User& user) {  // Code for sending notifications to the user  }  }; |

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

| **Principles(s):** Keep it simple, being able to check the arguments intended makes it easier to spot issues in the codebase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.STRUCT.DECL.FNEST** | Nested Function Declaration |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++1109, C++2510** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **CERT.DCL.AMBIGUOUS\_DECL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **296 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.

It is always immensely helpful to use automation throughout the development of software. From automated tests, to builds we can leverage automation to increase productivity and accuracy. However, it is very important that automation is used responsibly, or it could have the opposite effect. When use in the correct way we can use automation throughout the entire software development cycle.

### 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 | **P4** | **L3** |
| STD-002-CPP | Low | Unlikely | Medium | **P2** | **L3** |
| STD-003-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-004-CPP | Low | Unlikely | Medium | **P2** | **L3** |
| STD-005-CPP | High | Probable | High | **P6** | **L2** |
| STD-006-CPP | Low | Probable | Medium | **P4** | **L3** |
| STD-007-CPP | Low | Probable | Medium | **P4** | **L3** |
| STD-008-CPP | Low | Unlikely | Low | **P3** | **L3** |
| STD-009-CPP | Medium | Unlikely | Medium | **P4** | **L3** |
| STD-010-CPP | Low | Unlikely | Medium | **P2** | **L3** |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | The practice of encrypting the data when it is stored, for example sitting on a database. This applies depending on the sensitivity of the data and how the data is stored. |
| Encryption at flight | The practice of securing the data when its being transmitted over networks or other channels. This is important to prevent data leaks even if the data is intercepted when traveling over the network or other channels. |
| Encryption in use | Encrypting sensitive data while it is actively being processed or used by applications or systems. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Refers to the process of verifying the identity of a person, system, or entities. |
| Authorization | After user are authenticated, authorization determines what actions or resources an authenticated entities can use or access. |
| Accounting | Keeps track of and logging user activities, system events, and access attempts |

**\***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 | 6/18/2023 | [Insert text.] | Fabian Rodriguez | Fabian Rodriguez |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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