**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 | The process where all data received from outside sources is evaluated to see if it meets a specific criteria, such as format, length, and type, before it is processed. Validating the input data can help to prevent vulnerabilities such as SQL injections and buffer overflows. |
| 1. Heed Compiler Warnings | Warnings that indicate potential problems or vulnerabilities within the code that should be addressed as they appear. These messages may not prevent the code from compiling, but they should still be addressed promptly to prevent further problems. |
| 1. Architect and Design for Security Policies | When security is added throughout during the design of the software. When security is added during the design phase it helps the developers meet the organization’s security policies and requirements. Architecture Design for Security Policies includes setting up a framework and policies, defining risks and preventing them, and incorporating security controls. |
| 1. Keep It Simple | Complex is not always better. Keep it simple means to design software that meets all requirements but is as simple as possible so that errors and vulnerabilities may be avoided. A simple design will allow developers to design, test, maintain, and continue to keep secure because the system will be easier to navigate and detect problems. |
| 1. Default Deny | This process is where all access to a system is denied unless initially granted permission. By denying all incoming traffic that was not permitted, we limit the risk of access from unauthorized users and decrease the risk of vulnerabilities from happening. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege means that each user, process, or system is permitted the minimum privileges necessary to perform their given tasks. By giving each user, process, or system the minimum privilege we can prevent damage from compromised accounts and prevent attacks from escalating due to having to much access in the system. |
| 1. Sanitize Data Sent to Other Systems | The process where a source is sanitized by removing or encoding that may be considered harmful to the receiving party before sending them the data. Sanitizing data before sending it to other systems helps to prevent errors in formatting and injection attacks. |
| 1. Practice Defense in Depth | A process where multiple layers of security are used throughout an application to help prevent attackers from successfully attacking. Each security measure has their own loophole for attackers to breach. By using multiple different layers of security, you are helping to avoid those loopholes that attackers may find. This way if an attacker makes it though a layer of security, there is another layer followed to prevent them from attacking successfully. |
| 1. Use Effective Quality Assurance Techniques | The process where continuous analysis, testing, feedback, and audits are performed to help detect vulnerabilities early on. This process is continuously used so that the product, service, or process meets the requirements and standards. |
| 1. Adopt a Secure Coding Standard | Standards such as those provided by CERT and OWASP that help developers perform best practices and prevent vulnerabilities when developing. These standards provide developers with a set of guidelines so that they can write safe code that can stan up against attacks. |

### 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 non-compliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Do Not Declare or Define a Reserved Identifier |

**Source:** <https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL51-CPP.+Do+not+declare+or+define+a+reserved+identifier>

| **Noncompliant Code** |
| --- |
| The code uses a reserved name as the header guard which may clash with reserved names defined by the implementation of the C++ standard template library in its header or with reserved names implicitly predefined by the compiler even when no C++ standard library is included. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| To fix this the solution is to avoid using leading or trailing underscores in the name of the header guard. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

| **Principles(s):**  2. Heed Compiler Warnings: Compilers will often warn when reserved identifiers are misused.  10. Adopt a Secure Coding Standard: Reserved identifiers are restricted by the language standard. Following secure coding standards will help to avoid namespace conflicts and undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **reserved-identifier** | Partially checked |
| [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. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL51-a** **CERT\_CPP-DCL51-b** **CERT\_CPP-DCL51-c** **CERT\_CPP-DCL51-d** **CERT\_CPP-DCL51-e** **CERT\_CPP-DCL51-f** | Do not #define or #undef identifiers with names which start with underscore Do not redefine reserved words Do not #define nor #undef identifier 'defined' The names of standard library macros, objects and functions shall not be reused The names of standard library macros, objects and functions shall not be reused (C90) The names of standard library macros, objects and functions shall not be reused (C99) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **reserved-identifier** | Partially checked |

#### Coding Standard 2

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

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

| **Noncompliant Code** |
| --- |
| In this code an uninitialized local variable is evaluated as part of an expression to print its value. Because of this the outcome results in undefined behavior. |
| #include <iostream>    **void** f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| The solution to this is to initialize the object before printing out its value. |
| #include <iostream>    **void** f() {  **int** i=0;    std::cout << i;  } |

| **Principles(s):**  4.Keep It Simple: A simpler initialization logic reduces the risk of leaving variables uninitialized.  9. Use Effective Quality Assurance Techniques: Fuzz testing and memory analysis can detect access to uninitialized memory. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wuninitialized clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-EXP53-a** | Avoid use before initialization |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **LANG.STRUCT.RPL LANG.MEM.UVAR** | Return pointer to local Uninitialized variable |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: EXP53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp53cpp.html) | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Guarantee that storage for strings has sufficient space for character data and null terminator |

**Source:** <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator>

| **Noncompliant Code** |
| --- |
| In this example the unformatted input function std::basic\_istream<T>::read() is used to read an unformatted character array of 32 characters from the given file. However, the string being null terminated is not guaranteed by the read() function. Because of this the subsequent call std::string constructor will result in undefined behavior if a null terminator is not in the character array. |
| #include <fstream>  #include <string>    **void** f(std::istream &in) {  **char** buffer[32];  **try** {      in.read(buffer, **sizeof**(buffer));    } **catch** (std::ios\_base::failure &e) {      // Handle error    }      std::string str(buffer);    // ...  } |

| **Compliant Code** |
| --- |
| In this example the code is assuming that the input from the file is at most 32 characters. To fix the noncompliant statement we read the number of characters from the input stream to construct the std::string object. Unlike before where a null terminator was inserted. If the size is not certain then it is best to use std::basic\_istream<T>::readsome() or a formatted input function, whichever is needed for that specific situation. |
| #include <fstream>  #include <string>    **void** f(std::istream &in) {  **char** buffer[32];  **try** {      in.read(buffer, **sizeof**(buffer));    } **catch** (std::ios\_base::failure &e) {      // Handle error    }    std::string str(buffer, in.gcount());    // ...  } |

| **Principles(s):**  1. Validate Input Data – Ensures that input data is not longer than expected, which will prevent buffer overflow.  5. Default Deny – Enforces strict memory access and only permits defined and safe memory usage. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| |  | | --- |   [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stream-input-char-array** | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **MISC.MEM.NTERM LANG.MEM.BO LANG.MEM.TO** | No space for null terminator Buffer overrun Type overrun |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 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' |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stream-input-char-array** | Partially checked |

#### Coding Standard 4

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

**Source:** <https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection>

| **Noncompliant Code** |
| --- |
| In this example the code concentrates the user input directly into the SQL query string. This method is not safe and an attacker could use malicious SQL code to manipulate the query. |
| #include <iostream>  #include <string>  #include <sqlite3.h>  void vulnerableLogin(sqlite3\* db) {  std::string username, password;  std::cout << "Username: ";  std::cin >> username;  std::cout << "Password: ";  std::cin >> password;  // Dangerous: directly concatenating user input into SQL query string  std::string sql = "SELECT \* FROM users WHERE username = '" + username +  "' AND password = '" + password + "';";  char\* errMsg = nullptr;  int rc = sqlite3\_exec(db, sql.c\_str(), nullptr, nullptr, &errMsg);  if (rc != SQLITE\_OK) {  std::cerr << "SQL error: " << errMsg << std::endl;  sqlite3\_free(errMsg);  } else {  std::cout << "Query executed (but possibly vulnerable!)" << std::endl;  }  } |

| **Compliant Code** |
| --- |
| To fix the non-compliant code we can use prepared statements. By using this method, we can prevent SQL attacks by separating the code from the data. |
| #include <iostream>  #include <string>  #include <sqlite3.h>  void safeLogin(sqlite3\* db) {  std::string username, password;  std::cout << "Username: ";  std::cin >> username;  std::cout << "Password: ";  std::cin >> password;  const char\* sql = "SELECT \* FROM users WHERE username = ? AND password = ?;";  sqlite3\_stmt\* stmt = nullptr;  // Prepare statement  if (sqlite3\_prepare\_v2(db, sql, -1, &stmt, nullptr) != SQLITE\_OK) {  std::cerr << "Failed to prepare statement" << std::endl;  return;  }  // Bind parameters safely (prevents injection)  sqlite3\_bind\_text(stmt, 1, username.c\_str(), -1, SQLITE\_STATIC);  sqlite3\_bind\_text(stmt, 2, password.c\_str(), -1, SQLITE\_STATIC);  // Execute  int rc = sqlite3\_step(stmt);  if (rc == SQLITE\_ROW) {  std::cout << "Login successful!" << std::endl;  } else {  std::cout << "Invalid username or password." << std::endl;  }  sqlite3\_finalize(stmt);  } |

| **Principles(s):**  1. Validate Input Data – Validating user input at entry reduces risk of injection.  7. Sanitize Data Sent to Other Systems – Sanitization or parameterization of data before database use directly prevents injection.  8. Practice Defense in Depth – Combine input validation, prepared statements, and role-based access for layered protection. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | Likely | Medium | Critical | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/java/CodeSonar) | 9.0p0 | **JAVA.IO.INJ.SQL** | SQL injection |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2024.2 | **CERT.IDS00.TDSQL** | Protect against SQL injection |

#### Coding Standard 5

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

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

| **Noncompliant Code** |
| --- |
| In this example S is dereferenced after it has been deallocated. The vulnerability may be exploited to run arbritrary code with the permission of the vulnerable process if the access results in a write-after -free. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| To fix the code we can make it to where 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;  } |

| **Principles(s):**  6. Adhere to the Principle of Least Privilege – Proper privilege separation helps limit the damage from memory corruption.  9. Use Effective Quality Assurance Techniques – Memory sanitization tools can help detect use-after-free errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Critical | Likely | Medium | Critical | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **dangling\_pointer\_use** | - |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | |  | | --- | |  |  |  | | --- | | Checked by clang-tidy, but does  not catch all violations of this rule. | |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | |  | | --- | |  |  |  | | --- | | **USE\_AFTER\_FREE** | | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Do not abruptly terminate the program |

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

| **Noncompliant Code** |
| --- |
| In this example the call to f(), which is registerd as an exit handler with std::at\_exit(), might result to a call to std::terminate() because the throwing\_finc() may throw an exception. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| To fix this issue we set the call of f() to handle all exceptions thrown by the throwing\_func() and it will not rethrow. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.  **try** {      throwing\_func();    } **catch** (...) {      // Handle error    }  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

| **Principles(s):**  4. Keep It Simple – Simple, well-defined shutdown or error-handling logic prevents unexpected termination.  8. Practice Defense in Depth – Graceful failure and recovery mechanisms are additional layers of protection. |
| --- |

**Threat Level**

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

**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) | 9.1p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: ERR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr50cpp.html) | Checks for implicit call to terminate() function (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stdlib-use** | Partially checked |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle all exceptions |

**Source:** <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

| **Noncompliant Code** |
| --- |
| In this example, neither f() or main() catch exceptions that are thrown by throwing\_func(). Therefore std::terminate() is called because no matching handler can be found for the exception thrown. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| The solution to this is to set the main entry point to handle all exceptions. This will ensure that the stack is unwound up to the main() function and will allow for graceful management of the external resources. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {  **try** {      f();    } **catch** (...) {      // Handle error    }  } |

| **Principles(s):**  3. Architect and Design for Security Policies – Proper exception handling requires thoughtful architecture to ensure all failure points are managed.  8. Practice Defense in Depth – Catching and managing exceptions adds a layer of protection against crashes or code exposure. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Possible | Low | High | 4 |

**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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 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 |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Security Reviewer - Static Reviewer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Security+Reviewer+-+Static+Reviewer) | 6.02 | **C35** | Fully implemented |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Expressions** | STD-008-CPP | Do not cast or delete pointers to incomplete classes |

**Source:** <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP57-CPP.+Do+not+cast+or+delete+pointers+to+incomplete+classes>

| **Noncompliant Code** |
| --- |
| In this example the class Handle attempts to implement the impl idiom but deletes a pointer to an incomplete class type. This results in undefined behavior if Body has a nontrivial destructor. |
| **class** Handle {  **class** Body \*impl;  // Declaration of a pointer to an incomplete class  **public**:    ~Handle() { **delete** impl; } // Deletion of pointer to an incomplete class    // ...  }; |

| **Compliant Code** |
| --- |
| To fix this problem, we move the deletion of impl to a part of the code where Body is defined. |
| **class** Handle {  **class** Body \*impl;  // Declaration of a pointer to an incomplete class  **public**:    ~Handle();    // ...  };    // Elsewhere  **class** Body { /\* ... \*/ };    Handle::~Handle() {  **delete** impl;  } |

| **Principles(s):**  2. Heed Compiler Warnings – Compilers and static analyzers often flag unsafe pointer operations.  10. Adopt a Secure Coding Standard – Type-safe design and correct memory operations are a key part of secure coding. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Possible | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **delete-with-incomplete-type** | - |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | |  | | --- | |  |  |  | | --- | |  |   -Wdelete-incomplete | - |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **LANG.CAST.PC.INC PARSE.DOIC** | Conversion: pointer to incomplete delete of incomplete class |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: EXP57-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp57cpp.html) | Checks for conversion or deletion of incomplete class pointer |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | STD-009-CPP | Use valid references, pointers, and iterators to reference elements of a container |

**Source:** <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR51-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+container>

| **Noncompliant Code** |
| --- |
| In this example pos is invalidated after the first call to insert(), and subsequent loop iterations have undefined behavior. |
| #include <deque>    **void** f(**const** **double** \*items, std::**size\_t** count) {    std::deque<**double**> d;    auto pos = d.begin();  **for** (std::**size\_t** i = 0; i < count; ++i, ++pos) {      d.insert(pos, items[i] + 41.0);    }  } |

| **Compliant Code** |
| --- |
| The solution to this problem is to assign pos to a valid iterator on each insertion. This will prevent undefined behavior. |
| #include <deque>    **void** f(**const** **double** \*items, std::**size\_t** count) {    std::deque<**double**> d;    auto pos = d.begin();  **for** (std::**size\_t** i = 0; i < count; ++i, ++pos) {      pos = d.insert(pos, items[i] + 41.0);    }  } |

| **Principles(s):**  4. Keep It Simple – Overly complex iterator logic leads to bugs and instability.  9. Use Effective Quality Assurance Techniques – Unit and boundary testing help catch invalid iterators and memory issues. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **overflow\_upon\_dereference** | - |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **ALLOC.UAF** | Use After Free |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-CTR51-a** | Do not modify container while iterating over it |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024b | CERT C++: CTR51-CPP | Checks for use of invalid iterator (rule partially covered). |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data | STD-010-CPP | Obey the one-definition rule |

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

| **Noncompliant Code** |
| --- |
| In this noncompliant code, there are two different translation units defining a class of the same name but with differing definitions. Even though the two definitions are functionally equivalent, meaning they both define a class named S with a single, public, nonstatic data member int a, they are not defined using the same sequence of tokens. This is an example where the code violates the ODR and results in undefined behavior. |
| // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |

| **Compliant Code** |
| --- |
| To correct this code, it will depend on the intent of the programmer. In this example the programmer intends to use the same class definition to be visible in both translation units because of common usage. The solution to this is to use a header file to introduce the object into both translation units. |
| // S.h  **struct** S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

| **Principles(s):**  3. Architect and Design for Security Policies – A modular and well-designed system architecture avoids multiple definitions across components.  10. Adopt a Secure Coding Standard – Following language rules like ODR avoids linker issues and undefined behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Possible | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 9.1p0 | **LANG.STRUCT.DEF.FDH LANG.STRUCT.DEF.ODH** | Function defined in header file Object defined in header file |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2024.2 | **CERT\_CPP-DCL60-a** | The One Definition Rule shall not be violated |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024b | [CERT C++: DCL60-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl60cpp.html) | Checks for inline constraints not respected (rule partially 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.

In order for DevOps to become DevSecOps, each step in the diagram must be modified to incorporate security measures. In the “Asses and Plan” phase automation can be used to analyze threat models, assess regulatory changes, and evaluate risk. Tools that can be used include ThreatModeler or Microsoft Threat Modeling Tool. These tools will ensure that developers proactively plan for secure design. During the “Design” phase tools can be used to automate static code analysis against OWASP and other secure coding best practices. For the “Build” phase CI pipelines integrate automated security checks. Tools such as Snyk can scan dependencies for known vulnerabilities. In the “Verify ans Test” phase Dynamic Application Security Testing with tools such as OWASP, static code reviews with SAST and SCA, and where applicable automated penetration testing to ensure vulnerability scanning, compliance testing, and functional validation.

Next when transitioning to production in the “Transition and Health Check” phase automated tools should check configuration settings and deploy only if they meet policy requirements. In the “Monitor and Detect” phase automation enables real-time monitoring and alerting. The tools used can identify breaches and log security events. Next in the “Respond” phase security incident automation will help to isolate, mitigate, or roll back changes quickly. This can ensure rapid and consistent response. Lastly In the “Maintain and Stabilize” phase continuous security validation is performed. Automated tools are used in this phase to automatically remediate or alert on deviations from secure configurations. This will ensure long-term compliance and policy enforcement. By integrating automation into the DevSecOps infrastructure it will allow Green Pace to maintain a secure software development lifecycle that policy-compliant.

### 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-006-CPP | Medium | Possible | Low | Medium | 3 |
| STD-008-CPP | Medium | Possible | Medium | Medium | 3 |
| STD-002-CPP | High | Likely | Low | High | 4 |
| STD-003-CPP | High | Likely | Medium | High | 4 |
| STD-007-CPP | High | Possible | Low | High | 4 |
| STD-009-CPP | High | Likely | Low | High | 4 |
| STD-010-CPP | High | Possible | Medium | High | 4 |
| STD-004-CPP | Critical | Likely | Medium | Critical | 5 |
| STD-005-CPP | Critical | Likely | Medium | Critical | 5 |

### 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 | Encryption at rest protects stored data from unauthorized access. This may be data stored in databases, disks, and backups. This applies because it significantly reduces the risk of data breaches and ensures compliance with regulations such as HIPAA, GDPR, and CCPA. |
| Encryption in flight | Encryption in flight protects data transmitted across internal and external networks. The policy states all data must be encrypted using strong transport protocols such as TLS 1.2 or higher. This applies because it ensures data confidentiality and integrity when traversing public or untrusted networks. |
| Encryption in use | Encryption in use protects data that is considered in-use. This includes data that is being created, edited, or actively being processed in memory. This type of data is a big target for attackers. This policy applies because protecting data in use helps reduce emerging threats and supports secure cloud computing and zero trust architectures. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication ensures that users and systems are who they say to be or confirms one’s identity. Authentication is used to enforce MFA for user logins, use token-based authentication for APIs and services, and to integrate with identity providers for centralized user management. The reason this applies is because using strong authentication prevents unauthorized access and is essential for secure user access and regulatory compliance. |
| Authorization | Authorization is where users and systems are granted minimal required permissions based on their role. Access will be denied unless explicitly allowed. Authorization is applied by implementing role-based access control for all systems, restrict administrative actions to limited users, and uses automated checks to validate that new users and database changes follow access control policies. This policy applies by limiting access, which minimizes the impact of compromised accounts and also ensures compliance with security best practices. |
| Accounting | Accounting is where all system and user activities, login, access to data, changes in user roles, and modifications to databases must be logged so that it can be reviewed. Accounting helps detect, investigate, and respond to malicious activity by ensuring traceability. This applies because it aids in audits for regulatory reporting and operational transparency. |

**\***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 | 07/20/2025 | Module 3 Milestone | Rodrey McCoin |  |
| 1.2 | 08/10/2025 | Module 6 Project One | Rodrey McCoin |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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