**Green Pace Developer: Security Policy Guide**



# 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 proper testing of input provided by a user or application. Validation of the input prevents poorly formatted data from entering an information system. Such invalid inputs can be subject to injection attacks, memory leakage, or compromised systems. |
| 1. Heed Compiler Warnings | Use highest warning level possible when compiling code. This will reveal potential risks in current code logic. Use the warnings and analysis tools to eliminate all additional security risks. |
| 1. Architect and Design for Security Policies | Software architectures designed to implement and enforce security policies. Ensuring systems are proper set up to reduce unauthorized access to data. Examples include separating a system into distinct subsystems, each set with their specific privileges. |
| 1. Keep It Simple | Ensuring design is as simple and small as possible. Complex designs increase the inherent risk of security flaws. Making design simple ensures risks are easier to detect and fix. |
| 1. Default Deny | Access protocol based on permissions rather than exclusion. This means access is broadly denied to all users unless specifically designated. This ensures only those who need to access data will be able to do so. Reducing potential unauthorized incursions. |
| 1. Adhere to the Principle of Least Privilege | All processes should execute with the lowest level privileges possible for a given task. Only granting higher level privileges when necessary and only for the time required to complete the higher-level privilege task. This reduces the chances for attackers to gain access to higher level privileges that can be used to execute arbitrary code. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed between complex systems. Ensuring data passed between systems are not taking advantage of subsystems design flaws. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies. If one layer proves inadequate to prevent attacks another layer of defense may prove to be able to stop a successful exploitation. |
| 1. Use Effective Quality Assurance Techniques | Policy of testing and analyzing code base for potential threats. This reduces flaws and ensures a consistent basis for testing systems for flaws. Overall risk is reduced and enables the development of potentially superior security systems. |
| 1. Adopt a Secure Coding Standard | The use of standards when developing code that are proven to reduce security risk. Provides the first layer of defense against potential intrusions. |

### 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** | **Ensure unsigned integer operations do not wrap** |
| --- | --- | --- |
| **Data Type** | INT-030-C | If using an unsigned integer value, it is crucial that those values do not exceed the limit of the integer type assigned. Unaddressed wrapping can cause exploitable vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The Following example can result in an unsigned integer wrap during the addition of the unsigned operands ui\_a and ui\_b. Addition of this sort can potentially return a value that may be used to allocate insufficient memory or may lead to an exploitable vulnerability. |
| **void** func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum = ui\_a + ui\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution performs a precondition test of the operands of addition to ensure there is no unsigned wrap. |
| #include <limits.h>    **void** func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum;  **if** (UINT\_MAX - ui\_a < ui\_b) {      /\* Handle error \*/    } **else** {      usum = ui\_a + ui\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** ValidateInput Data – Wrapping occurs when input values exceed the limit specified by the data type |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **Integer-overflow** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) | 7.2.0 | **CertC-INT30** | Implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **INTEGER\_OVERFLOW** | Implemented |

Automation for this policy should be performed at the verification level of DevOps. Due to the nature of the error, it may become hard to locate or reproduce once in a production environment. The creation stage would also serve as a good place to make such automation checks. If both levels of development practice this automation strategy the likely hood of such an error propagating will be significantly reduced. Also reducing time needed to refactor the program to resolve any issue.

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that division and remainder operations do not result in divide-by-zero errors** |
| --- | --- | --- |
| **Data Value** | INT-033-C | Failing to follow standard can result in abnormal termination and denial of service |

| **Noncompliant Code** |
| --- |
| The code presented does not properly check for potential zero values in the division operation. Lack of quality checking can result in a divide by zero operation occurring. |
| #include <limits.h>    **void** func(**signed** **long** s\_a, **signed** **long** s\_b) {  **signed** **long** result;  **if** ((s\_a == LONG\_MIN) && (s\_b == -1)) {      /\* Handle error \*/    } **else** {      result = s\_a / s\_b;    }    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution tests the decision operation to ensure there is no possibly of divide-by-zero or signed overflow. It does this by checking for a zero value in the first if statement. Resulting in an error being thrown. |
| #include <limits.h>    **void** func(**signed** **long** s\_a, **signed** **long** s\_b) {  **signed** **long** result;  **if** ((s\_b == 0) || ((s\_a == LONG\_MIN) && (s\_b == -1))) {      /\* Handle error \*/    } **else** {      result = s\_a / s\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** Valid Input Data – Failure is the result of bad data being passed to a division operator, checking this value beforehand can help prevent bugs |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | **P6** | **L2** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **zerodiv zerodivcond** | Context sensitive analysis of division by zero Not detected for division by struct member / array element / pointer data that is 0 Detected when there is unsafe division by variable before/after test if variable is zero |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **DIVIDE\_BY\_ZERO** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-INT33-a** | Avoid division by zero |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule INT33-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint33c.html) | Checks for:   * Integer division by zero * Tainted division operand * Tainted modulo operand   Rule fully covered. |

#### 

#### Providing automation for this policy would be best placed in the Prepod phase. Due to the nature of the error, many inputs may be required to verify the existence of such an error. Attempts can also be made at previous phases such as creation and verify to decrease the odds of such an error. However, validation checking should occur with fuzz testing numerous inputs to determine the existence of such an error.

sw

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | STR-050-CPP | Copying data to a buffer that is not large enough to hold such data results in a buffer overflow. |

| **Noncompliant Code** |
| --- |
| The following code can lead to a buffer overflow due to the fact input is unbounded. The buffer can only hold 12 ‘char’ elements but the input stream places no limit on how much data can be passed to this buffer. |
| #include <iostream>    **void** f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The following compliant solution uses ‘std::string’ instead of a bounded buffer to store input. This will prevent truncation and guards against buffer overflow ensuring the string is passed correctly. |
| #include <iostream>  #include <string>    **void** f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):**  1.) Validate input Data: Validation input data to ensure it fits with in the provided buffer  3.) Architect & Design: Ensuring sound design principles can help prevent designing easily overflowable  7.) Sanitize Data: Vetting data entering the buffer can help prevent malicious attacks for the memory |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.2p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **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' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation   Rule partially covered. |

Automation for this policy should be focused on the phases of creation and Prepod. In the creation phase considerations should be made to ensure data can not easily produce overflows for buffer space. Adding additional automation in the Prepod phase will catch potential flaws in the initial design. Allowing the development team to resolve errors before being pushed to further operations, reducing time and cost.

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL injection** |
| --- | --- | --- |
| **SQL Injection** | IDS -000-J | Prevents unauthorized access to data outside of privileged search attempts. |

| **Noncompliant Code** |
| --- |
| The following example permits SQL injection attacks by incorporating the unsanitized input argument ‘username’ into the prepared statement. |
| **class** Login {  **public** Connection getConnection() **throws** SQLException {      DriverManager.registerDriver(**new**              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  **return** DriverManager.getConnection(dbConnection);    }      String hashPassword(**char**[] password) {      // Create hash of password    }    **public** **void** doPrivilegedAction(      String username, **char**[] password    ) **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String pwd = hashPassword(password);        String sqlString = "select \* from db\_user where username=" +          username + " and password =" + pwd;        PreparedStatement stmt = connection.prepareStatement(sqlString);          ResultSet rs = stmt.executeQuery();  **if** (!rs.next()) {  **throw** **new** SecurityException("User name or password incorrect");        }          // Authenticated; proceed      } **finally** {  **try** {          connection.close();        } **catch** (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses a parametric query using the ‘?’ character as a placeholder for the argument. It also validates the length of username, preventing an attacker form submitting an arbitrarily long suer name. |
| **public** **void** doPrivilegedAction(    String username, **char**[] password  ) **throws** SQLException {    Connection connection = getConnection();  **if** (connection == **null**) {      // Handle error    }  **try** {      String pwd = hashPassword(password);        // Validate username length  **if** (username.length() > 8) {        // Handle error      }        String sqlString =        "select \* from db\_user where username=? and password=?";      PreparedStatement stmt = connection.prepareStatement(sqlString);      stmt.setString(1, username);      stmt.setString(2, pwd);      ResultSet rs = stmt.executeQuery();  **if** (!rs.next()) {  **throw** **new** SecurityException("User name or password incorrect");      }        // Authenticated; proceed    } **finally** {  **try** {        connection.close();      } **catch** (SQLException x) {        // Forward to handler      }    }  } |

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

| **Principles(s):**  6.) Least privilege: Ensuring that breaches don’t have access to the entire system helps prevent serious breaches  7.) Sanitize Data: Vetting data being sent into the system prevents unauthorized use of common breach techniques  3.) Architecture and Design: First step of preventing such breaches is to design a robust Architecture preventing entry |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2021.2 | **CERT.IDS00.TDSQL** | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 6.7 | [**S2077**](https://rules.sonarsource.com/java/RSPEC-2077)  [**S3649**](https://rules.sonarsource.com/java/RSPEC-3649) | [Executing SQL queries is security-sensitive](https://rules.sonarsource.com/java/RSPEC-2077)  [SQL queries should not be vulnerable to injection attacks](https://rules.sonarsource.com/java/RSPEC-3649) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |

#### Automation against such a threat should be placed in the Prepod and Detection stage. The Prepod phase will help test for certain breach attempts that can then be fixed. Detection will be the active source of security. Ensuring any attempts at breaches are caught early or prevented altogether.

#### Coding Standard 5

| **Coding Standard** | **Label** | **Free dynamically allocated memory when no longer needed** |
| --- | --- | --- |
| **Memory Protection** | MEM-031-C | Failing to free memory can result in the exhaustion of system memory resources, potentially leading to a denial-of-service attack. |

| **Noncompliant Code** |
| --- |
| This noncompliant code does not free the object allocated by the call to ‘malloc()’ before the end of the lifetime of the last pointer ‘text\_buffer’. |
| #include <stdlib.h>    **enum** { BUFFER\_SIZE = 32 };    **int** f(**void**) {  **char** \*text\_buffer = (**char** \*)**malloc**(BUFFER\_SIZE);  **if** (text\_buffer == NULL) {  **return** -1;    }  **return** 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution deallocates the pointer with a call to ‘free()’ |
| #include <stdlib.h>    **enum** { BUFFER\_SIZE = 32 };    **int** f(**void**) {  **char** \*text\_buffer = (**char** \*)**malloc**(BUFFER\_SIZE);  **if** (text\_buffer == NULL) {  **return** -1;    }    **free**(text\_buffer);  **return** 0;  } |

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

| **Principles(s):**  10.) Coding Standards: Good practice in clearing memory when not in use prevents the error all together  2.) Compiler Warnings: Such warnings will help developers catch potential open memory  6.) Least privilege: Allowing only certain people to access data and interact with it can help reduce extra consequences of such an error |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM31** | Can detect dynamically allocated resources that are not freed |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **leakReturnValNotUsed** | Doesn't use return value of memory allocation function |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **429** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2021a | [CERT C: Rule MEM31-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem31c.html) | Checks for memory leak (rule fully covered) |

#### The create stage should be the place for Automation. This error is a result of developer error in freeing memory. Thus, providing the extra layer of checks to help developers catch such an error should greatly reduce error propagation. Additionally adding automation in the verify phase can help catch any potentially tricky leaks that the developer may not have caught. Reducing risk even further.

#### Coding Standard 6

| **Coding Standard** | **Label** | **Incorporate diagnostic tests using assertions** |
| --- | --- | --- |
| **Assertions** | MSC-011-C | Assertions help detect software defects that may result in vulnerabilities |

| **Noncompliant Code** |
| --- |
| This noncompliant example uses ‘assert()’ to verify memory allocation succeeded. Doing this can result in an abrupt termination of the process. Increasing the risk of denial-of-service attacks. |
| **char** \*dupstring(**const** **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char** \*)**malloc**(len + 1);  **assert**(NULL != dup);    **memcpy**(dup, c\_str, len + 1);  **return** dup;  } |

| **Compliant Code** |
| --- |
| This complaint example properly demonstrates how to detect and handle possible memory exhaustion. |
| **char** \*dupstring(**const** **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char**\*)**malloc**(len + 1);    /\* Detect and handle memory allocation error \*/  **if** (NULL == dup) {  **return** NULL;    }    **memcpy**(dup, c\_str, len + 1);  **return** dup;  } |

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

| **Principles(s):**  10.) Coding Standards: Adding additional checking to verify correctness of logic produces a more secure system  9.) Quality Assurance: quality testing the code base ensures logical errors are reduced  4.) Keep it Simple: Easy to understand logic is more manageable and less prone to unknown risks and easier testing |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **LANG.FUNCS.ASSERTS** | Not enough assertions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ASSERT\_SIDE\_EFFECT** | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-MSC11-a** | Assert liberally to document internal assumptions and invariants |

#### Automation should be placed in the plan, creation, and verify stages. Producing assurance testing should have its roots in the initial conception of a product or software. Ensuring logical coherence of the system and its intended logical effects will make development and testing of this logic simpler. This simplicity in design should produce positive results due to the structure. Lack of such structured assurance tests may result in complex logic that isn’t easy to test properly.

#### Coding Standard 7

| **Coding Standard** | **Label** | **Guarantee exception safety** |
| --- | --- | --- |
| **Exceptions** | ERR-056-CPP | Code that is not exception safe can lead to resource leaks, which will eventually result in undefined behavior after the first exception is thrown. |

| **Noncompliant Code** |
| --- |
| The following noncompliant example shows a flawed copy assignment operator. The ‘new’ expression throws an exception. Causing the object to be in an indeterminate state, resulting in undefined behavior. |
| **class** IntArray {  **int** \*array;    std::**size\_t** nElems;  **public**:    // ...      ~IntArray() {  **delete**[] array;    }        IntArray(**const** IntArray& that); // nontrivial copy constructor    IntArray& operator=(**const** IntArray &rhs) {  **if** (**this** != &rhs) {  **delete**[] array;        array = nullptr;        nElems = rhs.nElems;  **if** (nElems) {          array = **new** **int**[nElems];          std::**memcpy**(array, rhs.array, nElems \* **sizeof**(\*array));        }      }  **return** \***this**;    }      // ...  }; |

| **Compliant Code** |
| --- |
| This compliant example provides a strong exception safety guarantee. After the allocation succeeds the function proceeds to change the state of the object. Avoiding the test for self-assignment, improving the performance of the code. |
| **class** IntArray {  **int** \*array;    std::**size\_t** nElems;  **public**:    // ...      ~IntArray() {  **delete**[] array;    }      IntArray(**const** IntArray& that); // nontrivial copy constructor      IntArray& operator=(**const** IntArray &rhs) {  **int** \*tmp = nullptr;  **if** (rhs.nElems) {        tmp = **new** **int**[rhs.nElems];        std::**memcpy**(tmp, rhs.array, rhs.nElems \* **sizeof**(\*array));      }  **delete**[] array;      array = tmp;      nElems = rhs.nElems;  **return** \***this**;    }      // ...  }; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.2p0 | **ALLOC.LEAK** | Leak |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S, 56 D, 71 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-ERR56-a CERT\_CPP-ERR56-b** | Always catch exceptions Do not leave 'catch' blocks empty |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not use an additive operator on an iterator if the result would overflow** |
| --- | --- | --- |
| **Buffer Overflow** | CTR-055-CPP | Failing to do so will result in an overflow, causing data to be lost or unauthorized access to undesired memory locations. |

| **Noncompliant Code** |
| --- |
| This noncompliant example uses a random access iterator in an additive expression. The resulting value could be outside the bounds of the container rather than a past-the-end value. |
| **void** f(**const** std::vector<**int**> &c) {  **for** (auto i = c.begin(), e = i + 20; i != e; ++i) {      std::cout << \*i << std::endl;    }  } |

| **Compliant Code** |
| --- |
| This compliant solution assumes that the programmer intends to process up to 20 items in the container. The size of the container is used to determine the upper bound on addition, dynamically preventing overflow. |
| **void** f(**const** std::vector<**int**> &c) {  **const** std::vector<**int**>::size\_type maxSize = 20;  **for** (auto i = c.begin(), e = i + std::min(maxSize, c.size()); i != e; ++i) {      // ...    }  } |

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

| **Principles(s):**  1.) Valid input Data: Avoiding the use of improper data manipulation prevents bad data being passed  4.) Keep it Simple: Complex logic can be prone to errors such as a single variable exceeding its limit |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++3526, C++3527, C++3528, C++3529, C++3530, C++3531, C++3532, C++3533, C++3534** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **567 S** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-CTR55-a** | Do not add or subtract a constant with a value greater than one from an iterator |

Automation should be placed in the creation phase of development. Such an error is often generated in the initial stages of developing logic. Providing resources to the developer to catch this error, will reduce overall risk and cost. Later stages may also find this flaw difficult to catch one the program has reached its full state. Such errors are simpler to identify before the complexity of the code based is build around it. Due top the nature of the mistake future logic may become dependent on its existence making refactoring more difficult to manage in later stages.

#### Coding Standard 9

| **Coding Standard** | **Label** | **Close files when they are no longer needed** |
| --- | --- | --- |
| **File Security** | FIO-051-CPP | Failing to properly close files can allow attackers to exhaust system resources and increases the risk that data written into in-memory file buffers will not be flushed in the event of an abnormal program termination. |

| **Noncompliant Code** |
| --- |
| This noncompliant example constructs a file object. Destructors are not called and the std::basic\_filebuf<T> object is not properly closed. |
| **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| In this compliant solution ‘std::fstream::close()’ is called before ‘std::terminate()’, ensuring the file resource is properly closed. |
| **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...    file.close();  **if** (file.fail()) {      // Handle error    }    std::terminate();  } |

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

| **Principles(s):**  5.) Default Deny: ensuring privileged roles are only able to access data  6.) least Privileges: Reduces the ability for intruders to access unsecure files  3.) Architecture and design: separating data files makes managing potential intrusions easier to detect and manage |
| --- |

**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) | 6.2p0 | **ALLOC.LEAK** | Leak |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

Automation should be placed in the prevent phase. At this stage tool should provide tooling for verification and other such authentication procedures to ensure access to files are prevented. In addition, any open files can be caught and closed before they are moved to the detect phase.

#### Coding Standard 10

| **Coding Standard** | **Label** | **Copy operations must not mutate the source object** |
| --- | --- | --- |
| **Object Oriented Programming** | OOP-058-CPP | Copy operations that mutate the source can lead to unexpected program behavior. |

| **Noncompliant Code** |
| --- |
| This noncompliant example copies the ‘A’, causing a mutation in the source operand by resetting the member variable ‘m’ to 0. When ‘std::fill()’ is called, the first element is copied having the original value of ‘obj.m ’. Which sets obj.m to 0. Subsequent calls will all retain the value 0. |
| **class** A {  **mutable** **int** m;    **public**:    A() : m(0) {}  **explicit** A(**int** m) : m(m) {}      A(**const** A &other) : m(other.m) {      other.m = 0;    }      A& operator=(**const** A &other) {  **if** (&other != **this**) {        m = other.m;        other.m = 0;      }  **return** \***this**;    }    **int** get\_m() **const** { **return** m; }  };    **void** f() {    std::vector<A> v{10};    A obj(12);    std::fill(v.begin(), v.end(), obj);  } |

| **Compliant Code** |
| --- |
| This complaint example copies ‘A’ without mutating the source. Ensuring that the vector contains equal copies of ‘obj’. A is also given operations that perform mutation when it is safe t do so. |
| **class** A {  **int** m;    **public**:    A() : m(0) {}  **explicit** A(**int** m) : m(m) {}      A(**const** A &other) : m(other.m) {}    A(A &&other) : m(other.m) { other.m = 0; }      A& operator=(**const** A &other) {  **if** (&other != **this**) {        m = other.m;      }  **return** \***this**;    }      A& operator=(A &&other) {      m = other.m;      other.m = 0;  **return** \***this**;    }    **int** get\_m() **const** { **return** m; }  };    **void** f() {    std::vector<A> v{10};    A obj(12);    std::fill(v.begin(), v.end(), obj);  } |

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

| **Principles(s):**  10.) Secure Coding Standard: Strong practices help prevent such errors  4.) keep it simple: Complexity when using objects can often lead too hard to untangle knots, simplicity is a friend |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-OOP58-a** | Copy operations must not mutate the source object |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: OOP58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop58cpp.html) | Checks for copy operation modifying source operand (rule partially covered) |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++4075** | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2021.4 | [**CERT.OOP.COPY\_MUTATES**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) | N/A |

Automation should be placed in the creation stage of development. This is due to how this error is produced. Catching such an error early in the process prevents additional logic being built on top of the error, which could result in numerus refactoring’s. Tooling should be able to provide developers the ability to check source objects maintain that state. As such errors will later be difficult to locate.

### 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.

[Insert your written explanations here.]

### 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 |
| IDS -000-J | High | Probable | Medium | **P12** | **L1** |
| MSC-011-C | Low | Unlikely | High | **P1** | **L3** |
| MEM-031-C | Medium | Probable | Medium | **P8** | **L2** |
| INT-030-C | High | Likely | High | P9 | L2 |
| INT-033-C | Low | Likely | Medium | **P6** | **L2** |
| STR-050-CPP | High | Likely | Medium | **P18** | **L1** |
| FIO-051-CPP | Medium | Unlikely | Medium | **P4** | **L3** |
| CTR-055-CPP | High | Likely | Medium | **P18** | **L1** |
| ERR-056-CPP | High | Likely | High | **P9** | **L2** |
| OOP-058-CPP | Low | Likely | Low | **P9** | **L2** |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is the practice of [preventing access to data by attackers using data encryption on the disk. It should be used to ensure access to data in the system requires additional security requirements such as the key to access all data in the system. |
| Encryption at flight | Encryption at flight is the practice of encryption when data is being transmitted. This ensures that if an attacker seeks to interrupt data transfers must require the key to decrypt any data. |
| Encryption in use | Encryption in use is the practice of securing any data currently in use. This supports encryption on data as it is being accessed or transmitted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the practice of determining who is attempting to access the system. This ensures only those designated to access data can do so. |
| Authorization | Authorization is the practice of ensuring only certain individuals have access to parts of a system. Promoting a secure data system that has distributive properties that make it more difficult for attackers to gain access to full system data. |
| Accounting | Accounting is the practice of logging and monitoring behavior on a system. This practice ensuring no unusual behavior or access occurs that is unauthorized. Helping catch potential breaches. |

**\***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.

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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