**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Green Pace Secure Development Policy 1](#_Toc1436279280)

[Contents 1](#_Toc914097252)

[Overview 3](#_Toc579315262)

[Purpose 4](#_Toc1706679013)

[Scope 4](#_Toc1697141554)

[Module Three Milestone 4](#_Toc451991870)

[Ten Core Security Principles 4](#_Toc1042808852)

[C/C++ Ten Coding Standards 5](#_Toc2066286198)

[Coding Standard 1 5](#_Toc1689140841)

[Coding Standard 2 7](#_Toc587163654)

[Coding Standard 3 9](#_Toc1086615603)

[Coding Standard 4 12](#_Toc1329138987)

[Coding Standard 5 16](#_Toc2100594014)

[Coding Standard 6 18](#_Toc2068029964)

[Coding Standard 7 20](#_Toc1968210226)

[Coding Standard 8 23](#_Toc638918912)

[Coding Standard 9 25](#_Toc2120374747)

[Coding Standard 10 27](#_Toc790143990)

[Defense-in-Depth Illustration 29](#_Toc433951162)

[Project One 30](#_Toc2144998014)

[Revise the C/C++ Standards 30](#_Toc1065200076)

[Risk Assessment 30](#_Toc661243753)

[Automated Detection 30](#_Toc1732292265)

[Automation 30](#_Toc1414639989)

[Summary of Risk Assessments 31](#_Toc1354477138)

[Create Policies for Encryption and Triple A 32](#_Toc1158967613)

[Map the Principles 34](#_Toc1895684409)

[Audit Controls and Management 35](#_Toc1251489533)

[Enforcement 35](#_Toc1763055139)

[Exceptions Process 35](#_Toc920071904)

[Distribution 35](#_Toc2081143615)

[Policy Change Control 36](#_Toc693385993)

[Policy Version History 36](#_Toc330261161)

[Appendix A Lookups 36](#_Toc1321956894)

[Approved C/C++ Language Acronyms 36](#_Toc1431894959)

## 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. Validate Input Data | All data entering the system should be marked as unsafe by default and should be checked. Thus, make sure input variables are only allowing for proper data types and ranges to be entered. Essentially, make sure to prevent malicious data from being entered for input. |
| 1. Heed Compiler Warnings | Compiler warnings exist to notify developers that there may be a vulnerability somewhere in the code. Even though the code may compile since they are just warnings, it would be best to make sure all warnings are gone before the code is deemed safe to execute. |
| 1. Architect and Design for Security Policies | No matter what, when we develop software, we must develop it with the intent to keep the software and its users safe. This includes using proper secure coding practices and tools to ensure the software is safe to use. |
| 1. Keep It Simple | When doing many things in life, the motto KISS, or keep it simple stupid is very useful because it makes it so we do not overcomplicate things. Regarding software, the motto KISS can be applied as well. Keeping our design simple and efficient will make it easier to prevent vulnerabilities or flaws in the software. |
| 1. Default Deny | By default, we will want to deny everybody access to the system and its functionalities. The only way for users to gain access to the system and its functions is by being given permissions through a verified method. This can help bolster the security of the software. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege essentially states that each user should be given the least amount of permissions/privileges when using the system. Users of our software should have the least amount of privileges required in order to accomplish what they need from the system. |
| 1. Sanitize Data Sent to Other Systems | When data is being sent from our system, we need to ensure it is checked and cleansed before it is sent. This means we need to have built-in checks in the code that determine if a piece of data is safe to send. Doing so will ensure that data sent from our system cannot be used against us. |
| 1. Practice Defense in Depth | When developing software, we need to create multiple layers of security to ensure our software and its users are safe. Our software needs to be designed so that if one layer of security is breached, then there are still several others that are securing the software. |
| 1. Use Effective Quality Assurance Techniques | When testing our software, we want to make sure that it works properly no matter what inputs are provided, or what sequences of functions are used in the software. Some quality assurance techniques that should be utilized are unit testing, integration testing (both of which can be automated using certain libraries), penetration testing, and live user tests. Also, having an external review of the software would be beneficial. |
| 1. Adopt a Secure Coding Standard | Make sure to employ coding standards for the specific programming language being used, as well as general coding standards such as proper documentation or proper workflow. Use updated, trusted libraries and dependencies as well. |

### 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] | Obey the one-definition rule which basically means to have an object or function only have one definition for an entire piece of software. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, two different translation units define a class of the same name with differing definitions. Although the two definitions are functionally equivalent (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 code example violates the ODR and results in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| The correct mitigation depends on programmer intent. If the ODR violation was a result of accidental name collision, the best mitigation solution is to ensure that both class definitions are unique, as in this compliant solution. |
| // a.cpp  **namespace** {  **struct** S {  **int** a;  };  }    // b.cpp  **namespace** {  **class** S {  **public**:  **int** a;  };  } |

**Source:** [DCL60-CPP. Obey the one-definition rule - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule)

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

| **Principles(s):** This standard will help us ensure that our system does not have pieces of code that perform the same function or functions that result in undefined or unknown behaviors. We need to make sure our coding is simple to avoid recreating already implemented functions or developing useless functions that could lead to exploits to the system. (Keep It Simple - 4) This will promote cleanliness and efficiency in the code. (Architect and Design for Security Policies - 3) |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | LANG.STRUCT.DEF.FDH  LAND.STRUCT.DEF.ODH | Function defined in header file  Object defined in header file |
| LDRA Tool Suite | 9.7.1 | 286 S, 287 S | Fully implemented |
| Astree | 22.10 | type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CCC] | Ensure that operations on signed integers do not cause an overflow. |

|  |
| --- |
| **Noncompliant Code** |
| This noncompliant code example can result in a signed integer overflow during the addition of the signed operands si\_a and si\_b |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

|  |
| --- |
| **Compliant Code** |
| This compliant solution ensures that the addition operation cannot overflow, regardless of representation |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||  ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  /\* ... \*/  } |

**Source:** [INT32-C. Ensure that operations on signed integers do not result in overflow - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow)

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

| **Principles(s):** Follow the compiler warnings when they pop up regarding overflows and int variables. Otherwise, leaving unsigned integers free without being checked could lead to undefined behavior. (Heed Compiler Warnings - 2) Design the code with the idea of making signed integers should not overflow. Use statements to validate that the integers will not overflow. (Validate Input Data - 1) |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | 9 / Medium to High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | Integer-overflow | Full checked |
| CodeSonar | 7.3p0 | 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 |
| Parasoft C/C++test | 2022.2 | CERT\_C-INT32-a CERT\_C-INT32-b CERT\_C-INT32-c | Avoid integer overflows Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |
| LDRA tool Suite | 9.7.1 | 493 S, 494 S | Partially implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CCC] | Make sure string variables do not overflow due to improper input and buffer overflow. |

| **Noncompliant Code** |
| --- |
| The gets() function reads characters from the stdin into a destination array until end-of-file is encountered or a newline character is read. Any newline character is discarded, and a null character is written immediately after the last character read into the array. |
| #include <stdio.h>    #define BUFFER\_SIZE 1024    void func(void) {  char buf[BUFFER\_SIZE];  if (gets(buf) == NULL) {  /\* Handle error \*/  }  } |

| **Compliant Code** |
| --- |
| The gets\_s() function reads, at most, one less than the number of characters specified from the stream pointed to by stdin into an array. If end-of-file is encountered and no characters have been read into the destination array, or if a read error occurs during the operation, then the first character in the destination array is set to the null character and the other elements of the array take unspecified values. |
| #define \_\_STDC\_WANT\_LIB\_EXT1\_\_ 1  #include <stdio.h>    enum { BUFFERSIZE = 32 };    void func(void) {  char buf[BUFFERSIZE];    if (gets\_s(buf, sizeof(buf)) == NULL) {  /\* Handle error \*/  }  } |

**Source:** [STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator)

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

| **Principles(s):** Make sure we follow the standard of validating input for strings before we accept them as input. I (Validate Input Data - 1) Also, ensure that the buffer size for strings is large enough, otherwise, buffer overflows could occur and lead to undefined or unwanted behavior. Basically, build code that ensures the size of the buffer is large enough to handle any string entered. (Architect and Design for Security Policies - 3) Make sure to always distrust data coming into the software to prevent security breaches. In a sense, always deny the input at the start before allowing it to be used. (Default Deny – 5) |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | 18 / High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | N/A | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data |
| CodeSonar | 7.3p0 | LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\* | Buffer overrun Type overrun No space for null terminator A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| LDRA tool suite | 9.7.1 | 489 S, 109 D, 66 X, 70 X, 71 X | Partially implemented |
| Polyspace Bug Finder | R2023a | CERT C: Rule STR31-C | 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 * Insufficient destination buffer size   Rule partially covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAV] | Prevent SQL Injection for any type of input. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows JDBC code to authenticate a user to a system. The password is passed as a char array, the database connection is created, and then the passwords are hashed.  Unfortunately, this code example permits a SQL injection attack by incorporating the unsanitized input argument username into the SQL command, allowing an attacker to inject validuser' OR '1'='1. The password argument cannot be used to attack this program because it is passed to the hashPassword() function, which also [sanitizes](https://wiki.sei.cmu.edu/confluence/display/java/Rule+BB.+Glossary#RuleBB.Glossary-sanitize) the input. |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.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 + "'";  Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery(sqlString);    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 with a ? character as a placeholder for the argument. This code also validates the length of the username argument, preventing an attacker from submitting an arbitrarily long user 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  }  }  } |

**Source:** [IDS00-J. Prevent SQL injection - SEI CERT Oracle Coding Standard for Java - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection)

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

| **Principles(s):** We need to ensure our data in the system is protected by utilizing proper anti-SQL injection techniques. This includes input validation and sanitizing all data coming into and out of the system. (Validate Input Data - 1) We do not want any data that leaves the system to be unchecked. We need to know where the data is going, and who it is traveling to. This will help us prevent attackers from gaining info through our system that could give them an advantage in attacking the system because the code would be cleaned before being sent out. (Sanitize Data Sent to Other Systems - 7) Also, make sure users are limited in what they can do to prevent them from entering malicious input. Limit their capabilities on what they can enter as an extra level of defense. This will require users to have roles that allow for only the necessary permissions. (Build with the intention that users do not need to be on an admin account to use the application) (Adhere to the Principle of Least Privilege - 6) |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium to High | 12 / High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 6.7 | S2077  S3649 | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |
| SpotBugs | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | Implemented |
|  | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CCC] | Free dynamically allocated memory when no longer needed. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the object allocated by the call to malloc() is not freed before the end of the lifetime of the last pointer text\_buffer referring to the object. |
| #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** |
| --- |
| In this compliant solution, the pointer is deallocated 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;  } |

**Source:** [MEM31-C. Free dynamically allocated memory when no longer needed - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/MEM31-C.+Free+dynamically+allocated+memory+when+no+longer+needed)

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

| **Principles(s):** Ensure that memory locations are freed before the end of execution or beyond their use. This should be considered from the start of development so it can be avoided in the most crucial times of the development process. (Adopt a Secure Coding Standard - 10) Making sure we free up memory locations after they are not needed anymore will also help prevent the system’s memory from being exhausted quickly. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | 8 / Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | N/A | Supported, but no explicit checker |
| CodeSonar | 7.3p0 | ALOOC.LEAK | Leak |
| Cppcheck | 1.66 | leakReturnValNotUsed | Doesn’t use return value of memory allocation function |
| LDRA tool suite | 9.7.1 | 50 D | Partially implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CCC] | Use static assertions to test the values of constant expressions. |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. Although the use of the runtime assertion is better than nothing, it needs to be placed in a function and executed. This means that it is usually far away from the definition of the actual structure to which it refers. The diagnostic occurs only at runtime and only if the code path containing the assertion is executed. |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| Static assertions allow incorrect assumptions to be diagnosed at compile time instead of resulting in a silent malfunction or runtime error. Because the assertion is performed at compile time, no runtime cost in space or time is incurred. An assertion can be used at file or block scope, and failure results in a meaningful and informative diagnostic error message. |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int),  "Structure must not have any padding"); |

**Source:** [DCL03-C. Use a static assertion to test the value of a constant expression - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression)

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

| **Principles(s):** We must utilize all the tools available to ensure expressions and functions within our code work properly. This includes using different testing mechanisms in many scenarios where it is necessary. (Use Effective Quality Assurance Techniques - 9) Using the proper testing tools, including assertions, will allow for us to find and eliminate code errors or exploits much quicker than by basic code review. Thus, it is important to make it a habit of always having proper testing mechanisms being used throughout the SDLC. (Adopt a Secure Coding Standard - 10) |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | 1 / Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Guarantee exception safety. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a flawed copy assignment operator. The function deallocates array and assigns the element counter, nElems, before allocating a new block of memory for the copy. As a result, if the new expression throws an exception, the function will have modified the state of both member variables in a way that violates the implicit invariants of the class. |
| #include <cstring>    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** |
| --- |
| In this compliant solution, the copy assignment operator provides the [strong exception safety](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-strongexceptionsafety) guarantee. The function allocates new storage for the copy before changing the state of the object. Only after the allocation succeeds does the function proceed to change the state of the object. |
| #include <cstring>    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;  }    // ...  }; |

**Source:** [ERR56-CPP. Guarantee exception safety - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR56-CPP.+Guarantee+exception+safety)

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

| **Principles(s):** Exception safety is necessary for a program or application to operate in the right way. We need to ensure that our errors are being handled properly, otherwise, problems could result with random error codes being thrown at a user when an error occurs or other unexpected behavior Thus, we must write code that handles exceptions in a safe way instead of relying on error codes from the code. (Practice Defense in Depth - 8) We should attempt to keep our exception handling simple, but efficient. This will make it much easier to ensure our exceptions are handled correctly. Instead of writing clever code, we should write simple, but efficient code. (Keep it Simple - 4) |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | 9 / Medium to High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | ALLOC.LEAK | Leak |
| LDRA tool suite | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| PolySpace Bug Finder | R2023a | CERT C++: ERR56-CPP | Checks for exceptions violating class invariant (rule fully covered). |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Pointers/Iterators** | [STD-008-CPP] | Use valid references, pointers, and iterators to reference elements of a container. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pos is invalidated after the first call to insert(), and subsequent loop iterations have [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #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** |
| --- |
| In this compliant solution, pos is assigned a valid iterator on each insertion, preventing 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);  }  }v |

**Source:** [CTR51-CPP. Use valid references, pointers, and iterators to reference elements of a container - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR51-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+container)

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

| **Principles(s):** Ensure that valid, in bounds references are made to elements of a container. At times, when storing something like an iterator or pointer to an element in a container, problems can occur where the stored object becomes invalid due to the container being modified. We need to protect against this so that we prevent overflows and missing values in containers. Otherwise, there could be an error or overflow that occurs as a result. (Architect and Design for Security Policies - 3) Thus, we need to ensure that we only use valid pointers, references, or iterators for elements in a container like a list or dictionary. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | 6 / Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | overflow\_upon\_dereference | N/A |
| CodeSonar | 7.3p0 | ALLOC.UAF | Use After Free |
| Polyspace Bug Finder | 2022.2 | CERT\_CPP-CTR51-a | Do not modify container while iterating over it |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Errors** | [STD-009-CPP] | Detect and handle memory allocation errors. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an array of int is created using ::operator new[](std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because ::operator new[](std::size\_t) can throw an exception if the allocation fails, it could lead to [abnormal termination](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination) of the program. |
| #include <cstring>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new int[size];  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new (std::nothrow) int[size];  if (!copy) {  // Handle error  return;  }  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy;  } |

**Source:** [MEM52-CPP. Detect and handle memory allocation errors - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors)

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

| **Principles(s):** Do not allow for memory allocation errors to stall the operation of the application or hinder its performance. Design the code in a way that memory is allocated and deallocated in a simple and efficient way. (Adopt a Secure Coding Standard - 10) Ensure any memory allocation errors that do occur are handled accordingly in a way that prevents the system from being attacked. (Practice Defense in Depth - 8) |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | 18 / High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Polyspace Bug Finder | R2023a | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Handling Exceptions** | [STD-010-CPP] | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {  throwing\_func();  }    **int** main() {  **try** {  f();  } **catch** (...) {  // Handle error  }  } |

**Source:** [ERR51-CPP. Handle all exceptions - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions)

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

| **Principles(s):** Do not leave any exception unhandled. This could lead to problems down the line when an exception is not handled and results in application errors or crashes. Test for these exception handling statements as a precaution. (Use Effective Quality Assurance Techniques - 9) If exceptions are left unhandled, then problems related to DoS attacks could be a result. Thus, it is necessary to ensure all exceptions are handled, and handled correctly. It also helps that exception handling can act as an extra layer of defense against attackers. (Practice Defense in Depth - 8) |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | 4 / Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | main-function-catch-all  early-catch-all | Partially checked |
| CodeSonar | 7.3p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| PolySpace Bug Finder | R2023a | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| LDRA tool suite | 9.7.1 | 527 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.

Automation is something that can be implemented throughout most of the DevOps process. The first place that automation should be implemented is within the Verify and test phase of pre-production. Unit testing, integration testing, penetration testing, etc... can all be automated in some way, shape, or form. Thus, it would be wise to utilize tools that allow for test automation to be used. Unit testing should also be used and automated in the build phase of the DevOps process because it will facilitate the process of the applications development. Also, in the build phase of the pre-production process, automation should be used to check code files to see if the dependencies being used are up to date and have no, or very miniscule vulnerabilities. This includes running dependency checks on the code every once in a while. In the Monitor and Detect portion of the Production process, we must use a tool to automate the process of gathering audits on how the users are using the system. Then, in the Maintain and stabilize portion of the process, there needs to be more automation tools used to streamline the process of updating/maintenance the systems/applications. Automation testing should also be utilized in the Transition and health check portion of the DevOps cycle. This will make it much better to test whether code is working as it should be.

### 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 | High | 3 / Low | 3 |
| STD-002-CCC | High | Likely | High | 9 / Medium to High | 2 |
| STD-003-CCC | High | Likely | Medium | 18 / High | 1 |
| STD-004-JAV | High | Probable | Medium to High | 12 / High | 1 |
| STD-005-CCC | Medium | Probable | Medium | 8 / Medium | 2 |
| STD-006-CCC | Low | Unlikely | High | 1 / Low | 3 |
| STD-007-CPP | High | Likely | High | 9 / Medium to High | 2 |
| STD-008-CPP | High | Probable | High | 6 / Medium | 2 |
| STD-009-CPP | High | Likely | Medium | 18 / High | 1 |
| STD-010-CPP | Low | Probable | Medium | 4 / Low | 3 |

### 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 in rest is the process of securing data when it is not moving or being used. The process of securing involves encrypting the data with an algorithm, which makes the data hard to understand if someone does not have the key to decrypt it. Encryption at rest is necessary for data when it is at rest because it acts as a strong layer of defense for the data involved in a system. Even if someone gets into the system, encryption at rest makes it so they need to decrypt the files before they have something easily usable. This policy applies because it is essentially a part of Defense in Depth, and it is useful in DevSecOps. Some examples of encryption algorithms that can be used are SHA-256 or AES. Encryption at rest is used when the data is being stored in a database and is not being used. |
| Encryption at flight | Encryption in flight is pretty much the process of adding a layer of encryption to data while it is in transit. Basically, while data is being sent from one place to another, encryption is used to make it harder for outsiders to decipher the data being sent. Encryption in transit usually involves sending data through secure channels or pathways that have encryption on them. One of the most secure channels used for data transfer is TSL or SSL. Encryption in flight/transit is a very good way to ensure that data is being protected while it is being sent to your users/clients. Pretty much anybody working with software that is sending and receiving data from others needs to secure their transfers with this type of encryption. |
| Encryption in use | Encryption in use is pretty much encryption that is used whenever data is being used within an application. This is, again, another extra layer of defense that makes it harder for attackers to decipher data being used in the system that they should not be able to understand. Encryption in use is implemented by making it so memory is protected, aka, when data is being used or prepared for being used. Encryption is use is not just a way to add an extra layer of protection to the application, it also helps prevent memory related errors. Encryption in use is used when the data is being stored in memory for usage later or currently. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of identifying who users are and making sure they are who they say they are. Applications use authentication to confirm who users are to allow them access to the system and its functionalities. For most systems and applications that we will develop, authentication will be a necessity because it will help us verify what users are using said systems/applications. This will bolster our security because users will be required to verify/identify themselves to use the application. (People will be prevented from accessing the system/application if they are not of the correct identity) Will help prevent unverified users from getting into the system/application. We could apply authentication in our systems through a username, password, account process. |
| Authorization | Authorization is the process of giving permissions to users based upon their roles within an application or system. It is the act of giving someone the ability to perform some kind of function. Authorization will be a necessity in all our applications/systems because it allows us to verify that authenticated users have the correct permissions. Having authorization in our systems will make it so unauthorized users cannot perform certain actions with the system/application. This could help prevent malicious users from utilizing our applications in a devious way. We could apply authorization in our systems by implementing a role-based access system. This will create roles that users will be assigned once they have created an account that can be authenticated. |
| Accounting | Accounting is the action of determining what resources were accessed/used. This includes when they were accessed, who accessed them, and how they were accessed. Accounting makes it so we know who, when, and why in relation to our system being accessed. This is very helpful because it will allow us to keep track of interactions within our applications/systems. Accounting for system/application interactions will also make it so auditing is much easier when it is needed for some reason. There are many ways to keep track of interactions within our systems/applications. The best option would be to utilize at least more than one different way of tracking interactions. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 04/09/2023 | Updated Security Policy | Dominic Clapper |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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