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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_heading=h.gjdgxs)

[Purpose 2](#_heading=h.30j0zll)

[Scope 2](#_heading=h.1fob9te)

[Module Three Milestone 2](#_heading=h.3znysh7)

[Ten Core Security Principles 2](#_heading=h.4d34og8)

[C/C++ Ten Coding Standards 3](#_heading=h.2s8eyo1)

[Coding Standard 1 4](#_heading=h.3rdcrjn)

[Coding Standard 2 5](#_heading=h.26in1rg)

[Coding Standard 3 6](#_heading=h.lnxbz9)

[Coding Standard 4 7](#_heading=h.35nkun2)

[Coding Standard 5 8](#_heading=h.1ksv4uv)

[Coding Standard 6 9](#_heading=h.2jxsxqh)

[Coding Standard 7 10](#_heading=h.z337ya)

[Coding Standard 8 11](#_heading=h.28h4qwu)

[Coding Standard 9 13](#_heading=h.nmf14n)

[Coding Standard 10 14](#_heading=h.37m2jsg)

[Defense-in-Depth Illustration 15](#_heading=h.1mrcu09)

[Project One 15](#_heading=h.46r0co2)

[1.](#_heading=h.2lwamvv) Revise the C/C++ Standards 15

[2.](#_heading=h.111kx3o) Risk Assessment 15

[3.](#_heading=h.3l18frh) Automated Detection 15

[4.](#_heading=h.206ipza) Automation 15

[5.](#_heading=h.4k668n3) Summary of Risk Assessments 16

[6.](#_heading=h.2zbgiuw) Create Policies for Encryption and Triple A 16

[7.](#_heading=h.1egqt2p) Map the Principles 17

[Audit Controls and Management 18](#_heading=h.3ygebqi)

[Enforcement 18](#_heading=h.2dlolyb)

[Exceptions Process 18](#_heading=h.sqyw64)

[Distribution 19](#_heading=h.3cqmetx)

[Policy Change Control 19](#_heading=h.1rvwp1q)

[Policy Version History 19](#_heading=h.4bvk7pj)

[Appendix A Lookups 19](#_heading=h.2r0uhxc)

[Approved C/C++ Language Acronyms 19](#_heading=h.1664s55)

## 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 | Input validation ensures only properly formatted data enters an information system, preventing issues downstream such as malformed string inputs, buffer overflow, or other malformed data types. It should be done as soon as data is received from an external source. |
| 1. Heed Compiler Warnings | To ensure the safety of your code, it is recommended to use the highest warning level available in your compiler and address any warnings by modifying the code accordingly. Additionally, static and dynamic analysis tools should be utilized to identify and eliminate any potential security vulnerabilities. |
| 1. Architect and Design for Security Policies | When creating software, it is important to design with security policies in mind. To achieve this, consider dividing the system into distinct subsystems that communicate with each other and have different privileges based on the specific needs of the system at different times. This will help to ensure that the appropriate privilege set is in place and enforced. |
| 1. Keep It Simple | It's best to keep the design simple and small to avoid errors during implementation, configuration, and use. As security mechanisms become more complex, the effort required to ensure their effectiveness also increases significantly. So, it's advisable to stick to uncomplicated designs. |
| 1. Default Deny | When deciding on Base access, it's important to prioritize permission over exclusion. This means that access should be automatically denied, and only granted under specific conditions determined by the protection scheme. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege entails limiting users' access rights to only what is strictly necessary for their job functions. Users are granted permission to access only the files or resources that are required for them to complete their tasks, and are restricted from unnecessary read, write, or execute privileges. |
| 1. Sanitize Data Sent to Other Systems | Data sanitization is the intentional and permanent removal or destruction of data from a storage device to prevent its recovery. Usually, when data is deleted from storage media, it's not completely erased and can still be retrieved by unauthorized persons. This poses a significant risk to data privacy and security. Through sanitization, the storage media is thoroughly cleaned, leaving no remaining data on the device, and making it impossible to recover any data, even with sophisticated forensic tools. |
| 1. Practice Defense in Depth | Defense in Depth involves implements several layers of defense mechanisms to safeguard valuable data and information. In case one mechanism fails, another one instantly takes over to prevent any attack. This approach, with its intentional redundant layers, provides heightened security to the entire system and effectively addresses various attack vectors. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance is a process that is an integral part of the quality management system and spans the entire life cycle of the product or service. Activities like product testing, process validation, and system integration may be involved in QA. |
| 1. Adopt a Secure Coding Standard | Developing secure code is crucial for creating high-quality software and protecting the organization from security threats. Writing code that is not secure can lead to negative consequences such as financial losses, reputational damage, market manipulation, theft, and more. Starting with secure coding practices ensures that applications have a solid foundation of security to safeguard the interests of businesses and their stakeholders. |

### 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** | **Implement abstract data types using opaque types** |
| --- | --- | --- |
| **Data Type** | [DCL-012-C] | Abstract data types allow for the combination of data and related operations into a unified unit of encapsulation. |

| **Noncompliant Code** |
| --- |
| Including the string\_m.h file allows full access to the string\_mx data type, which violates software engineering principles and can lead to incorrect or nonportable code. |
| **struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;  unsigned **char** strtype;  **char** \*cstr;  };    **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| The external string\_m.h file defines the string\_mx type as an instance of the struct string\_mx, which is made as an incomplete type. The internal header file fully defines the struct string\_mx, but it's not visible to users of the data abstraction. |
| **external string\_m.h file**  **struct** string\_mx;  **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* ... \*/  **Internal header file**  **struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;  unsigned **char** strtype;  **char** \*cstr;  }; |

**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** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | hgih | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL12 |  |
| LDRA | 9.7.1 | 104 D | The implementation is only partial. |
| Polyspace Bug Finder | R2023a | Cert C: Rec. DCL12-C | Verifies if a structure or union object implementation is present in a file where the object's pointer is not dereferenced. |
| Polyspace Bug Finder | 2023.1 | CERT\_C-DCL12-a | If a pointer to a structure or union is not dereferenced, its implementation should be hidden within a translation unit. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that integer conversions do not result in lost or misinterpreted data** |
| --- | --- | --- |
| **Data Value** | INT-031-C | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| The code uses two integers (ui\_a and ui\_b) and when the sum of the two exceeds the maximum value for an integer, wrapping occurs. This can result in the allocation of insufficient memory for other operations. |
| #include <limits.h>    **void** func(**void**) {  unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;  sc = (**signed** **char**)u\_a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Here's a code that can convert an unsigned long int value to a signed char value: |
| #include <limits.h>    **void** func(**void**) {  unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;  **if** (u\_a <= SCHAR\_MAX) {  sc = (**signed** **char**)u\_a; /\* Cast eliminates warning \*/  } **else** {  /\* Handle error \*/  }  } |

**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 | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | LANG.CAST.PC.AV LANG.CAST.PC.CONST2PTR LANG.CAST.PC.INT  LANG.CAST.COERCE LANG.CAST.VALUE  ALLOC.SIZE.TRUNC MISC.MEM.SIZE.TRUNC  LANG.MEM.TBA | Cast: arithmetic type/void pointer Conversion: integer constant to pointer Conversion: pointer/integer  Coercion alters value Cast alters value  Truncation of allocation size Truncation of size  Tainted buffer access |
| Coverity | 2017.07 | NEGATIVE\_RETURNS  REVERSE\_NEGATIVE  MISRA\_CAST | Can detect potentially dangerous integer conversions in array accesses, loop bounds, and other expressions. Also detects instances where a negative value is used before a negativity check or where an integer expression is implicitly converted to a narrower type. |
| CppCheck | 1.66 | memsetValueOutOfRange | There is an issue with the second argument of memset(), as it cannot be presented as an unsigned char. |
| LDRA tool suite | 9.7.1 | 93 S, 433 S, 434 S | The implementation is partial. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not pass a non-null-terminated character sequence to a library function that expects a string** |
| --- | --- | --- |
| **String Correctness** | [STR-032-C] | It is crucial to ensure that any string or wide string argument used in library functions is null-terminated. Failure to do so may result in accessing memory beyond the object's bounds if a non-null-terminated character sequence or wide character sequence is passed. |

| **Noncompliant Code** |
| --- |
| Note that the character sequence "c\_str" will not have a null termination when it is passed as an argument to printf(). |
| #include <stdio.h>    **void** func(**void**) {  **char** c\_str[3] = "abc";  **printf**("%s\n", c\_str);  } |

| **Compliant Code** |
| --- |
| When the array bound is left out, the compiler will allocate enough space to store the complete string literal, which including getting rid of the null character. |
| #include <stdio.h>    **void** func(**void**) {  **char** c\_str[] = "abc";  **printf**("%s\n", c\_str);  } |

**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 | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR32 | Partially implemented: able to detect certain rule violations. |
| CodeSonar | 7.4p0 | MISC.MEM.NTERM.CSTRING | not terminated C String |
| Coverity | 2017.07 | STRING\_NULL | Fully executed |
| Parasoft C/C++test | 2023.1 | CERT\_C-STR32-a | Checks for invalid use of standard library string functions  and tainted null or non-null-terminated strings. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL injection** |
| --- | --- | --- |
| **SQL Injection** | [IDS-000-J] | SQL injection vulnerabilities occur in applications when parts of a SQL query come from a source that cannot be trusted. If proper measures are not taken, the untrusted data can alter the query with malicious intent, causing data modification or information leaks. |

| **Noncompliant Code** |
| --- |
| The code below has a vulnerability that could allow for a SQL injection attack. This is due to the unsanitized input argument "username" being included in an SQL command, which could give an attacker the ability to inject "validuser" or "1=1". However, the "password" argument is not vulnerable to attack because it is passed through the "hashPassword()" function, which sanitizes 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** |
| --- |
| The code below utilizes a parametric query where a "?" character serves as a placeholder for the argument. This particular code ensures that the length of the username argument is valid, which effectively prevents an attacker from submitting a username that is excessively long. |
| **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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Errors related to trust and security. |
| CodeSonar | 7.4p0 | JAVA.IO.INJ.SQL | SQL Injection for Java |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Executed |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Executed |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | [MEM-050-CPP] | The decision to reallocate or recycle freed memory is up to the memory manager. When memory is freed, all pointers to it become invalid, and its contents may either be returned to the operating system and become inaccessible or remain accessible. This means that data at the freed location may appear valid but can change unexpectedly, so it's important not to write to or read from freed memory. |

| **Noncompliant Code** |
| --- |
| In the code below, there is an issue where the "s" is accessed after it has been deallocated. This can lead to a vulnerability where arbitrary code can be executed with the permissions of the vulnerable process if there is a write-after-free. Usually, it's challenging to identify and diagnose these problems because dynamic memory allocations and deallocations are usually far apart. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {  S \*s = **new** S;  // ...  **delete** s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| The allocated memory in this code is not being deallocated until it is no longer needed. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {  S \*s = **new** S;  // ...  s->f();  **delete** s;  } |

**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 | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checks by clang-tidy, but it does not catch all violations of the rule. |
| CodeSonar | 7.4p0 | ALLOC.UAF | Use after free |
| Coverity | v7.5.0 | USE\_AFTER\_FREE | It can detect when memory is deallocated multiple times or when a freed pointer is read/written to. |
| LDRA tool suite | 9.7.1 | 483 S, 484 S | The implementation is only partial. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | [DCL-003-C] | Using assertions is an effective way to diagnose and stop defects in software that could potentially lead to vulnerabilities. |

| **Noncompliant Code** |
| --- |
| This code is not compliant as it utilizes the assert() macro to declare a property about a memory-mapped structure that is critical for the correct functioning of the code. |
| #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** |
| --- |
| To handle assertions that only involve constant expressions, one can use a preprocessor conditional statement. |
| **struct** timer {  unsigned **char** MODE;  unsigned **int** DATA;  unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

**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** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | misc-static-assert | Inspected by clang-tidy |
| CodeSonar | 7.4p0 | (customization) | Users have the option to create a personalized check that detects instances of the assert() macro. |
| ECLAIR | 1.2 | CC2.DCL03 | Fully executed |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Honor exception specifications** |
| --- | --- | --- |
| **Exceptions** | [ERR-055-CPP] | If a function throws an exception that is not permitted by its exception-specification, it may cause the program to terminate in a way that is determined by the implementation. |

| **Noncompliant Code** |
| --- |
| The code declares a function as nonthrowing. However, it's possible for an exception to occur when std::vector::resize() fails to allocate the requested memory. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {  v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| This solution is compliant and removes the noexcept-specification from the function, indicating that the function permits all exceptions. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) {  v.resize(s); // May throw, but that is okay  } |

**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** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **unhandled-throw-noexcept** | the task gets partially checked |
| CodeSonar | 7.4p0 | **LANG.STRUCT.EXCP.THROW** | Uses throw |
| LDRA tool suite | 9.7.1 | 56 D | the task gets partially checked |
| Parasoft C/C++ Test | 2023.1 | **CERT\_CPP-ERR55-a** | When a function declares an exception-specification, it can only throw the specified exception(s). |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Use parentheses for precedence of operation** |
| --- | --- | --- |
| Expressions | [EXP-000-C] | Programers often make errors on precedence rules of C operators. Mistakes regarding precedence rules can be avoided by the suitable use of parentheses. Using parentheses defensively reduces errors and, if not taken to excess, makes the code more readable. |

| **Noncompliant Code** |
| --- |
| The purpose of the code's expression is to examine the least significant bit of x. |
| x & 1 == 0 |

| **Compliant Code** |
| --- |
| To ensure the expression evaluates as expected, parentheses are utilized in this code. |
| (x & 1) == 0 |

**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** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion | 7.2.0 | CertC-EXP00 | Fully executed |
| CodeSonar | 7.4p0 | LANG.STRUCT.PARENS | checks for missing parentheses |
| LDRA tool suite | 9.7.1 | 361 S, 49 S | Fully executed |
| Parasoft C/C++ test | 2023.1 | CERT\_C-EXP00-a | Use parentheses to clarify expression order when using operators with lower precedence than arithmetic. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not use std::rand() for generating pseudorandom numbers** |
| --- | --- | --- |
| Miscellaneous | [MSC-050-CPP] | Mathematical algorithms are employed by pseudorandom number generators to generate a sequence of numbers that possess favorable statistical characteristics; however, the resulting numbers are not truly random. |

| **Noncompliant Code** |
| --- |
| Here is some code that generates an ID using the rand() function to create a numeric part. However, it's important to note that these IDs are not entirely random and can be predicted to some extent. |
| #include <cstdlib>  #include <string>    **void** f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  id += std::to\_string(std::**rand**() % 10000);  // ...  } |

| **Compliant Code** |
| --- |
| The C++ standard library offers precise control over generating pseudorandom numbers. This process is divided into two parts. The first part is the engine, which is responsible for providing random values. The second part is the distribution, which ensures that values are accurately distributed within a specified range. Although the distribution object is not mandatory, it prevents bias issues in the random number generation process. |
| **void** f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  std::uniform\_int\_distribution<**int**> distribution(0, 10000);  std::random\_device rd;  std::mt19937 engine(rd());  id += std::to\_string(distribution(engine));  // ...  } |

**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** |
| --- | --- | --- | --- | --- |
| Medium | unlikely | low | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 4.0 | cert-msc50-cpp | Inspected by clang-tidy |
| CodeSonar | 7.4p0 | BADFUNC.RANDOM.RAND | Uses Random |
| Eclair | 1.2 | CC2.MSC30 | Fully executed |
| LDRA tool suite | 9.7.1 | 44 S | Improves enforcement measures. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Range check element access** |
| --- | --- | --- |
| [Student Choice] | [STR-053-CPP] | The const\_reference operator[](size\_type) and reference operator[](size\_type) of the std::string index return the character stored at a specific position, pos. If pos is greater than or equal to size(), a reference to an object of type charT with value charT() will be returned. These index operators do not check for errors in range and will not throw any exceptions. If an attempt is made to modify an out-of-range object, the behavior is undefined. |

| **Noncompliant Code** |
| --- |
| The code example provided is noncompliant and may result in undefined behavior. This is because the value returned by the get\_index() call could be greater than the total number of elements stored in the string. |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {  std::string s("01234567");  s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| This solution meets the required standards by utilizing the std::basic\_string::at() function. This function works similarly to the index operator[], but it will throw a std::out\_of\_range exception if the position (pos) is greater than or equal to the size of the string. |
| #include <stdexcept>  #include <string>  **extern** std::**size\_t** get\_index();    **void** f() {  std::string s("01234567");  **try** {  s.at(get\_index()) = '1';  } **catch** (std::out\_of\_range &) {  // Handle error  }  } |

**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 | Unlikely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Helix QAC | 2023.1 | C++3162,  C++3163,  C++3164,  C++3165 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP  -STR53-a | Ensure that the indices of the container are within the valid range. |
| Polyspace Bug Finder | R2023a | CERT C++: STR53-CPP | Checks for out-of-bounds array access, array access with tainted index, and pointer dereference with tainted offset. |

### 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 | Likely | Medium | P18 | L1 |
| [EXP-000-C] | Low | Probable | Medium | P4 | L3 |
| [DCL-003-C] | Low | Unlikely | High | P1 | L3 |
| [DCL-012-C] | Low | Unlikely | High | P1 | L3 |
| [INT-031-C] | High | Probable | High | P6 | L2 |
| [STR-032-C] | High | Probable | Medium | P12 | L1 |
| [MEM-050-CPP] | High | Likely | Medium | P18 | L1 |
| [MSC-050-CPP] | Medium | unlikely | low | P6 | L2 |
| [STR-053-CPP] | High | Unlikely | Medium | P6 | L2 |
| [ERR-055-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 | Making sure data is encrypted on a disk. This guideline ensures that if a hard drive is obtained by an attacker, they would be unable to access the data without the encryption keys. |
| Encryption at flight | Making sure data is encrypted during transmission is crucial. Often, people use vulnerable methods like Wi-Fi or cloud to move data, but encryption ensures security and prevents attackers from accessing the data. |
| Encryption in use | Making sure that data is encrypted at all stages is crucial. One of the ways this is achieved through the use of defense in depth (DID), which provides multiple layers of security. If a hacker manages to bypass one gateway, there are more layers they have to go through. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This policy requires users to authenticate before accessing sensitive data, allowing the system to accept or reject based on predefined criteria. An authentication system should be put in place as one of the first layers of defense. |
| Authorization | The next step is used to help determine whether the user is allowed to access specific resources and what operations can the user do. This is one of the first things in DID |
| Accounting | This step monitors activities and records actions for observance and protection purposes. This helps show risky behavior and patterns that can help prevent an attack. This is another layer within the DID. |

**\***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 |  |
| [Insert text.] | 07/23/2023 | Updated Coding Standards | Talia Cunningham | Professor Mill |
| [Insert text.] | 08/13/2023 | Updated everything | Talia Cunningham | Professor Mill |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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