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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | This principle attempts to eliminate as many vulnerabilities as possible by validating the data from all incoming sources, whether from a secured user or unknown source. In this way, malicious data can be found before anything bad can happen. |
| 1. Heed Compiler Warnings | Compile and test the code as much as possible, in this way, keeping the code tight and well written. To accomplish this, heeding all warnings is a requirement. A warning may seem innocent, but it could also mean something more serious is happening under the hood. |
| 1. Architect and Design for Security Policies | Building a system that has different subsystems that are accessed through privileges can help in enforcing security policies. By keeping people out of areas they aren’t supposed to be in from the onset, one can save time and money later on when a fix is needed. |
| 1. Keep It Simple | Keeping the code simple can help in multiple ways. First, clean and easy code can be streamlined, which helps when there are a lot of working/ moving parts. Also, clean, and simple code can be debugged much easier than, say, jumbles of code with bad syntax and poor use of while space. The alternative, complex code, has the chance of containing many errors that won’t be found until runtime. |
| 1. Default Deny | Access should be decided on permission and not exclusion. When the system is based around permission, access can be denied, but the user will then be directed to the proper channels to gain access. |
| 1. Adhere to the Principle of Least Privilege | Sets a standard of only those who need higher access can get it through the proper channels. This works well with Default Deny, because while going through the process of gaining access, it is expressly implemented that only the access that is needed is given. Also, the permission needed and accessed should be used for the minimal time that is required to complete the task. Following this standard will reduce a hacker’s chance at accessing privileged data. |
| 1. Sanitize Data Sent to Other Systems | All the data that is passed through any of the system’s subsystems must be sanitized. This means things such as command prompts/shells, relational databases, and other components, need to be free of sensitive data. Sending code with the data still present can lead to many exploits such as injections through the components, like SQL. |
| 1. Practice Defense in Depth | Most systems and subsystems require more than a single layer of protection. By combining many redundant layers of varying defense, the systems can be safeguarded. DiD protects the system if the first layer fails, preventing exploitable vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | QA is a massive part of any project. And by practicing QA techniques, you can identify and eliminate vulnerabilities in the system. Some of the techniques used include source code audits and penetration testing. |
| 1. Adopt a Secure Coding Standard | Having a Secure Coding Standard can go a long way in supporting all the other principles, not to mention designing your code with security policies built in from the onset. These policies can include a plan for QA, which can limit issues now and will save time and money down the road on potential fixes. |

### 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** | STD-001-GLC (DCL12) | Abstract data types are not restricted to object-oriented language such as C++ and Java. They should be created and used in C language programs as well. Abstract data types are most effective when used with private data types and information hiding. |

| **Noncompliant Code** |
| --- |
| The implementation of the string\_mx type is fully visible to the user of the data type after including the string\_m.h file. Programmers are consequently more likely to directly manipulate the fields within the structure, violating the software engineering principles of information hiding and data encapsulation and increasing the probability of developing 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 solution involves the String\_mx type to be declared as private, which in turn hides the implementation of the data type from the user. And the header file for struct string\_mx is completely invisible to user. |
| 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);  /\* ... \*/  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):** Architect and Design for Security policies & Adhere to the Principle of least privilege  – Making sure users can access only what is needed. This can be accomplished by making use of public, private and protected categories in the C++ code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL12 |  |
| LDRA tool suite | 9.7.1 | 104 D | Partially implemented |
| Polyspace Bug Finger | R2023a | Cert C: Rec. DCL12-C | Checks for structure or union object implementation visible in file where pointer to this object is not dereferenced. |
| Parasoft C/C++test | 2022.2 | CERT\_C-DCL12-a | If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do Not Use Floating-point Variables As Loop Counters** |
| --- | --- | --- |
| **Data Value** | [STD-002-C]  (FLP30) | Because floating-point numbers present real numbers, oftentimes people mistake their ability to represent any simple fraction exactly. But floating-point numbers are subject to some representational limitations just like integers, and binary-floating-point numbers cannot represent all real numbers exactly, even with a small number of decimal digits.  As such, floating-point variables should NOT be used as the loop induction variable. |

| **Noncompliant Code** |
| --- |
| Below, a floating-point variable is used as the counter for the loop. The decimal number is 0.1 is a repeating fraction in binary and cannot be exactly represented as a floating-point number. Because of this, the loop may run 9 times or 10, depending. |
| **void** func(**void**) {  **for** (**float** x = 0.1f; x <= 1.0f; x += 0.1f) {      /\* Loop may iterate 9 or 10 times \*/    }  } |

| **Compliant Code** |
| --- |
| The loop induction variable is an integer now. It count from 1 to 10, and will always reach ten loops. |
| #include <stddef.h>    **void** func(**void**) {  **for** (**size\_t** count = 1; count <= 10; ++count) {  **float** x = count / 10.0f;      /\* Loop iterates exactly 10 times \*/    }  } |

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

| **Principles(s):** Validate Input value – Using the correct variable type will lead to an expected outcome. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **for-loop-float** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-FLP30** | Fully implemented |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | cert-flp30-c | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.STRUCT.LOOP.FPC** | Float-typed loop counter |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do Not Attempt To Modify String Literals** |
| --- | --- | --- |
| **String Correctness** | [STD-003-C]  (STR30) | Avoid assigning a string literal to a pointer a non-const or casting a string literal to a pointer to a non-const |

| **Noncompliant Code** |
| --- |
| A char pointer is initialized to the address of a string literal. Then we try to modify the string literal to undefined results. |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| In this case, we use an array initializer for the string literal. From there, we are modify the string properly. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** Validate Input Data & Keep It Simple – Using the correct technique can solve errors before than occur. In this case, modifying the contents of an array after it has been stored returns correct values, unlike attempting to chance a literal stored in a pointer. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **string-literal-modfication** **write-to-string-literal** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-STR30** | Fully implemented |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | PW | Deprecates conversion from a string literal to "char \*" |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **C0556, C0752, C0753, C0754**  **C++3063, C++3064, C++3605, C++3606, C++3607** |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAV]  (IDS00-J) | SQL Injection vulnerabilities happen when elements of an SQL query originate from an untrusted source. Without having the proper precautions, the untrusted source can alter the query, causing information leaks or data modification. |

| **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. |
| **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        }      }    }  }  The JDBC library provides an API for building SQL commands that sanitize [untrusted data](https://wiki.sei.cmu.edu/confluence/display/java/Rule+BB.+Glossary#RuleBB.Glossary-untrusteddata). The java.sql.PreparedStatement class properly escapes input strings, preventing SQL injection when used correctly. This code example modifies the doPrivilegedAction() method to use a PreparedStatement instead of java.sql.Statement. However, the prepared statement still permits a SQL injection attack by incorporating the unsanitized input argument username into the prepared statement.   |  | | --- | | **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;        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 compliment solution uses a parametric query with a ? character as the placeholder for the argument. This code also validates the length of the username, which prevents any attacker from submitting an arbitrarily long user name to gain data. |
| **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):** Validate Input Value & Adhere to the Principle of Least Privilege & Practice DiD – Validating the user input to make sure a SQL-injection of the username or password cannot happen is key to stopping an attack before it can get worse. With an injection, hackers can possibly access vital information, which is why files should be placed in proper Public, Private, Protected sections, as another line of defense. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | 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) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Free Dynamically Allocated Memory When No Longer Needed** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C]  (MEM31) | Before the function has ended that uses the pointer that stores the return value, a call to free() with that pointer value must be matched. |

| **Noncompliant Code** |
| --- |
| In this 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 code, 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;  } |

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

| **Principles(s):** Heed Compiler Warnings – If memory is allocated for a pointer but it is not freed after the end of function, the code will compile but perhaps with a warning. It is important to heed the warning to call free() on the allocated memory to stop a memory leak before it happens. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 |  | Supported, but no explicit checker |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM31** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **ALLOC.LEAK** | Leak |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **RESOURCE\_LEAK**  **ALLOC\_FREE\_MISMATCH** | Finds resource leaks from variables that go out of scope while owning a resource |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use A Static Assertion To Test The Value Of A Constant Expression** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG]  (DCLO3) | Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities. The runtime assert() macro has some limitations, however, in that it incurs a runtime overhead and because it calls abort(). Consequently, the runtime assert() macro is useful only for identifying incorrect assumptions and not for runtime error checking. As a result, runtime assertions are generally unsuitable for server programs or embedded systems. |

| **Noncompliant Code** |
| --- |
| This code uses the assert() macro. It asserts a property that concerns a memory-mapped struct. This struct is essential for the code to work correctly |
| #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** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement can be used: |
| **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):** Heed Compiler Warnings & Adopt a Secure Coding Standard – With the aid of assertions, the code can be tested for vulnerabilities before it is packaged an shipped, helping to reduce time and money needed for fixing it after the fact. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Honor Exception Specifications** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP]  (ERR55) | If a function throws an exception other than one allowed by its exception-specification, it can lean to an implementation-defined termination of the program.  If a function declared with a dynamic-exception-specification throws an exception of the type that would not match the exception-specification, the function STD::unexpected() is called. The behavior of the function is overridden, but, by default, causes an exception of STD::bad\_exception to be thrown. Unless STD::bad\_exception is listed in the exception-specification, the function STD::terminate() will be called. |

| **Noncompliant Code** |
| --- |
| A function is declared as being non-throwing, but still, it IS possible for the std::vector::resize() to throw an exception when the requested allocated memory cannot be called. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {    v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| The function’s noexcept-specification is removed, signifying that the function allows 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):** Use effective quality assurance Techniques & Adopt a Secure Coding Standard – This standard makes use of unexpected() and bad\_exception() to be called when an exception other than the expected exception is thrown. In visual studio, for instance, micosoft VS provides \_declspec(nothrow). Throwing from a function declared with one of these language extensions is presumed undefinded behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **unhandled-throw-noexcept** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | 7.2.0 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.EXCP.THROW** | Use of throw |
| Use of throw | 2023.1 | **C++4035, C++4036, C++4632** |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Value-returning Functions must return a value from all exit paths** |
| --- | --- | --- |
| Data Value | [STD-008-CPP]  (MSC52) | A value-returning function must return a value from all code paths; otherwise, it will result in undefined behavior. This includes returning through less-common code paths, such as from a function-try-block. |

| **Noncompliant Code** |
| --- |
| In this function, the programmer forgot to return the input value for the positive input. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  } |

| **Compliant Code** |
| --- |
| The programmer made sure to cover positive and negative return values by adding a second return. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  **return** a;  } |

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

| **Principles(s):** Validate Input Data & Heed Compiler Warnings – Making sure to return a value from all portions of a function is vitally important to keep data whole and uncorrupted. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-implicit** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wreturn-type** | Does not catch all instances of this rule, such as function-try-blocks |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.MRS** | **LANG.STRUCT.MRS** |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Allocate sufficient memory for an object** |
| --- | --- | --- |
| Memory Protection | [STD-009-C]  (MEM35) | The types of integer expressions used as size arguments to malloc(), calloc(), realloc(), or aligned\_alloc() must have sufficient range to represent the size of the objects to be stored. If size arguments are incorrect or can be manipulated by an attacker, then a buffer overflow may occur. Incorrect size arguments, inadequate range checking, integer overglow, or truncation can result in the allocation of an inadequately sized buffer. |

| **Noncompliant Code** |
| --- |
| The inadequate space is allocated for the struct tm object because the size of the pointer is being used to determine the size of the pointed-to object. |
| #include <stdlib.h>  #include <time.h>    **struct** **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {  **struct** **tm** \*tmb;    tmb = (**struct** **tm** \*)**malloc**(**sizeof**(tmb));  **if** (tmb == NULL) {  **return** NULL;    }    \*tmb = (**struct** **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };  **return** tmb;  } |

| **Compliant Code** |
| --- |
| The correct amount of memory is allocated for the struct tm object. When allocating space for a single object, passing the (dereferenced) pointer type to the sizeof operator is a simple way to allocate sufficient memory. |
| #include <stdlib.h>  #include <time.h>    **struct** **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {  **struct** **tm** \*tmb;    tmb = (**struct** **tm** \*)**malloc**(**sizeof**(\*tmb));  **if** (tmb == NULL) {  **return** NULL;    }    \*tmb = (**struct** **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };  **return** tmb;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard & Architect and Design for Security Policies – Making sure to allocate sufficient memory for use in all functions can be the difference between a hacker overflowing a buffer and gaining access to privileged information. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | |  |  | | --- | --- | | **malloc-size-insufficient** |  | | Partially checked  Besides direct rule violations, all undefined behaviour resulting from invalid memory accesses is reported by Astrée. |
| |  |  | | --- | --- | | [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=125337650) |  | | 7.2.0 | **CertC-MEM35** |  |
| |  |  | | --- | --- | | [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) |  | | |  |  | | --- | --- | | 7.3p0 |  | | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW ALLOC.SIZE.TRUNC IO.TAINT.SIZE MISC.MEM.SIZE.BAD LANG.MEM.BO LANG.MEM.BU LANG.STRUCT.PARITH LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU** | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could check violations of this rule by examining the size expression to malloc() or memcpy() functions. Specifically, the size argument should be bounded by 0, SIZE\_MAX, and, unless it is a variable of type size\_t or rsize\_t, it should be bounds-checked before the malloc() call. If the argument is of the expression a\*b, then an appropriate check is   |  | | --- | | **if** (a < SIZE\_MAX / b && a > 0) ... | |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Clear sensitive information stored in reusable resources** |
| --- | --- | --- |
| Memory Protection | [STD-010-C]  (MEM03) | Sensitive data stored in reusable resources may be inadvertently leaked to a less privileged user or attacker if not properly cleared. Reusable resources include: Dynamically allocated memory, statically allocated memory, automatically allocated (stack) memory, memory caches, disk, disk caches. |

| **Noncompliant Code** |
| --- |
| Sensitive information stored in the dynamically allocated memory referenced by secret is copied to the dynamically allocated buffer, new\_secret, which is processed and eventually deallocated by a call to free(). Because the memory is not cleared, it may be reallocated to another section of the program where the information stored in the new\_seret may be unintentionally leaked. |
| **char** \*secret;  /\* Initialize secret to a null-terminated byte string,     of less than SIZE\_MAX chars \*/    **size\_t** size = **strlen**(secret);  **char** \*new\_secret;  new\_secret = (**char** \*)**malloc**(size+1);  **if** (!new\_secret) {    /\* Handle error \*/  }  **strcpy**(new\_secret, secret);    /\* Process new\_secret... \*/    **free**(new\_secret);  new\_secret = NULL; |

| **Compliant Code** |
| --- |
| Dynamic memory containing sensitive information should be sanitized before being freed. Sanitation is commonly accomplished by clearing the allocated space (that is, filling the space with ‘\0’ characters). |
| **char** \*secret;  /\* Initialize secret to a null-terminated byte string,     of less than SIZE\_MAX chars \*/    **size\_t** size = **strlen**(secret);  **char** \*new\_secret;  /\* Use calloc() to zero-out allocated space \*/  new\_secret = (**char** \*)**calloc**(size+1, **sizeof**(**char**));  **if** (!new\_secret) {    /\* Handle error \*/  }  **strcpy**(new\_secret, secret);    /\* Process new\_secret... \*/    /\* Sanitize memory \*/  memset\_s(new\_secret, '\0', size);  **free**(new\_secret);  new\_secret = NULL; |

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

| **Principles(s):** Sanitize Data Sent to Other Systems – If Data is not removed after its use has ended, while also being secured for use by those with privilege, the data is at ask risk of being hacked. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **(customization)** | Users can add a custom check for use of realloc(). |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect possible violations of this rule by first flagging any usage of realloc(). Also, it could flag any usage of free that is not preceded by code to clear out the preceding memory, using memset. This heuristic is imperfect because it flags all possible data leaks, not just leaks of "sensitive" data, because ROSE cannot tell which data is sensitive |
| |  |  | | --- | --- | | [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) |  | | 2023.1 | **C5010** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced Enforcement |

### 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.

Most importantly, we need to ensure that “Verify and Test” are our top priorities during the Pre-production phase of development. Having a functional, error-free product before production is key. Starting production with a product with major, or even minor, security flaws can be detrimental to the exposed system.

As for the production phase, transition and health checks should happen often and with great care. Using penetration tests can tell us soon and often if any holes or issues might have slipped through the cracks.

The above diagram describes the overall production of a product. In the pre-production phase, the team is planning, designing, and building the product while also checking for security faults along the way. The team focuses on finding and eradicating security vulnerabilities in the production phase. And the beauty is the process is not just a “one-time through” production. With repetition, the product is only strengthened and to the point of being shippable.

### 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 | Low | Unlikely | High | Low | 3 |
| STD-002-CPP | Low | Probable | Low | Medium | 2 |
| STD-003-CPP | Low | Likely | Low | Medium | 2 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | Medium | Probable | Medium | Medium | 2 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Likely | Low | Medium | 2 |
| STD-008-CPP | Medium | Probable | Medium | Medium | 2 |
| STD-009-CPP | High | Probable | High | Medium | 2 |
| STD-0010-CPP | Medium | Unlikely | High | 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 | The definition of encryption at rest is when data is encrypted when it is being stored. This means the data is still accessible as a readable type but can only be accessed with the proper key. Doing things this way applies better upfront defense, halting many in-person thefts because of the use of the key for accessing the data. |
| Encryption at flight | Unlike encryption at rest, when at flight means the data is being encrypted while transmitted. However, The data is not encrypted when it is being stored or while someone is using it, but instead while moving to another storage location. This technique is great when workers transfer data back and forth while working at separate locations. Even if the data is intercepted, it will still encrypt so long as the interceptor doesn’t have the correct access key. |
| Encryption in use | Encryption in use means just how it sounds: while an employee is using the data, it is also being encrypted. This gives certain users access to the data depending on their security access levels, applying another layer of defense. This policy prevents newer/lower-level employees from accessing the data or the system. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication uses passwords/codes, secure networks, usernames/logins, sometimes facial and fingerprint recognition, and two-feature identification depending on the user at the time and their security level. |
| Authorization | Authorization leverages the features of authentication to identify the correct level of access, if any, to give to the user currently logging in. For example, someone with the admin security level can access databases, files, records, financials, etc. They can also make changes, add and change users, and give certain data file privileges to other users. |
| Accounting | Accounting leverages features from both authentication and authorization to account for—or audit—the traffic on the system. In this way, there is a tangible way to track when data has been changed and by whom. Some systems even have the user making the change comment on what they are doing, which gets stored for later review. This framework adds yet another layer of defense. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 06/10/2023 | Total review | Dan Stull |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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