**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 functionologies 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 | It is important to validate all incoming data. This helps to lower the chances of vulnerabilities in the software. The biggest part to consider is that all data comes from somewhere and this helps to be aware of that while ensuring the data is safer. |
| 1. Heed Compiler Warnings | By paying attention to compiler warnings, the developer is being notified of potential errors and risks that can be found within the code. The importance of this is to allow the developer reduce holes in security and bugs. |
| 1. Architect and Design for Security Policies | Security is an important part of the architecture and design of a system, network, or program. By implementing security policies, and including them into the design, one is allowing for knowledge on who has access to what. It is a good idea for keeping different systems more self-contained and also addressing any possible risks before they become true risks. |
| 1. Keep It Simple | Keeping the code clean and simple helps in finding vulnerabilities and patching them up. When work gets a too complicated, errors, security holes, and bugs can appear more frequently. These can become difficult to manage or find when there’s more complexity. |
| 1. Default Deny | It is important to keep roles and permissions at a minimum. By having a default deny policy, all areas are denied access until certain conditions are met. This keeps the wrong people from being in areas that they are not supposed to be. |
| 1. Adhere to the Principle of Least Privilege | The principle of Least Privilege means that the users of the system are given the minimum amount of access needed for their job. This increases security because you’re not allowing users to do whatever they want. This is a good concept because it means the users are less likely to be opening holes that they should not. This relates to the concept of default deny as well. This is due to being denied access because they do not have the correct conditions to meet the requirements. |
| 1. Sanitize Data Sent to Other Systems | Data needs to be sanitized when transmitted between systems. This is a way to remove any content that can be potentially harmful. This helps in preventing SQL injection attacks and cross-site scripting. The idea is that the data is safe when processed and only the clean data is exchanged between the systems. |
| 1. Practice Defense in Depth | Practicing Defense in Depth is important because you’re adding layers of security to your system or network. This means that it isn’t the default and you’re increasing the level of security and time it takes for hackers, or malicious software, to get through the system. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques are varied. These often include security testing, authentication, integration testing, document quality, evidence-based decision making, continuous monitoring, automation, and close collaboration. These techniques are important in increasing the security and finding any vulnerabilities from within the system. This is done in multiple ways through early documents in the design, but also in real-time tests that find any vulnerabilities and report them before the areas are made available publicly. |
| 1. Adopt a Secure Coding Standard | Having a secure coding standard can help result in consistency. It helps ensure that there’s security already within the system, that users can’t do whatever they want, that when things fail, you can remain secure, and fix errors fast. There’s also ensuring that everything in the system is modular and capable of being a component that is more easily manageable. |

### 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** | **Obey the One-Definition Rule** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | While the majority of programs can contain multiple translation units for linking them together, there is a precedence in C++ to have the objects restricted in order to be contained by one definition.  “Every program shall contain exactly one definition of every non-inline function or variable that is odr-used in that program; no diagnostic required. The definition can appear explicitly in the program, it can be found in the standard or a user-defined library, or (when appropriate) it is implicitly-defined. An inline function shall be defined in every translation unit in which it is odr-used.” (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) |

Source, code examples, and other sections all comes from:

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

| **Noncompliant Code** |
| --- |
| The below example of noncompliant code is taking two translation units that are different, but using a class of the same name. The class has a different definition from the other translation unit. This is a violation when connecting to the ODR. |
| //a.cpp  struct S {  int a;  };  //b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| The below example of being compliant is by using a header to create the definition, in this case of an integer called a. It is then included in the other files to bring in the information without having to create units that are the same and different. This allows the translation units to be seen by the other files. |
| Struct S {  int a;  };  //a.cpp  #include “S.h”  //b.cpp  #include “S.h” |

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

| **Principles(s):**  *Architect and Design for Security Policies*: This allows for including security in the entire process of development. It does not become an afterthought. This keeps security as a focus of the process as well.  *Keep It Simple*: Keeping the system simple allows for more easily fixing any issues that could appear.  *Adopt a Secure Coding Standard*: The adoption of a secure standard allows for avoiding issues that could potentially open holes in a system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Hight | Unlikely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **“type-compatibility** **definition-duplicate** **undefined-extern** **undefined-extern-pure-virtual** **external-file-spreading** **type-file-spreading”** (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This has been partially checked. |
| Axivion Bauhause Suite | 7.2.0 | “CertC++-DCL60” (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| CodeSonar | 8.1p0 | “LANG.STRUCT.DEF.FDH  LANG.STRUCT.DEF.ODH” (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This tool does  “Function defined in header file Object defined in header file” (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| Helix QAC | 2024.3 | “C++1067, C++1509, C++1510” (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| LDRA Tool Suite | 9.7.7 | “286 S, 287 S” (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This tool is listed has having full implementation. |
| Parasoft C/C++test | 2023.1 | “CERT\_CPP-DCL60-a” (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) | Tool defines as “A class, union or enum name (including qualification, if any) shall be a unique identifier” (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| Polyspace Bug Finder | R2024a | “CERT C++: DCL60-CPP” (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This tool has checks in regards to inline constraints. It also is listed as not respect and partially covered. |
| RuleChecker | 22.10 | **“type-compatibility** **definition-duplicate** **undefined-extern** **undefined-extern-pure-virtual** **external-file-spreading** **type-file-spreading”** (*DCL60-CPP. Obey the One-definition Rule - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This has been partially checked. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not Read Uninitialized Memory** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Reading uninitialized memory can cause problems and C++ has a standard that creates a default initialization if the object has nothing to begin with. Reading this can result in issues when the default is used because there is an indeterminate value in place.  “As a result, objects of type T with automatic or dynamic storage duration must be explicitly initialized before having their value read as part of an expression unless T is a class type or an array thereof or is an unsigned narrow character type. If T is an unsigned narrow character type, it may be used to initialize an object of unsigned narrow character type, which results in both objects having an [indeterminate value](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-indeterminatevalue). This technique can be used to implement copy operations such as std::memcpy() without triggering [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior).” (*EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) |

Source, code examples, and other sections all comes from:

*EXP53-CPP. Do not read uninitialized memory - SEI CERT C++ Coding Standard - Confluence*. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory

| **Noncompliant Code** |
| --- |
| The example of noncompliant code shows a value that has not been initialized with any integer value. It is currently set to whatever the default is for C++. This causes issues when going to print as there is no value within the integer that is recognizable by the compiler for output. |
| #include <iostream>  Void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The example of compliant code includes the value of 0 being added after the creation of the variable. This allows for the compiler to read the base value set into the variable and print the value. This means that the compiler is capable of reading what is there. |
| #include <iostream>  void f() {  int i = 0;  std::out << i;  } |

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

| **Principles(s):**  *Validate Input Data*: having the data validated allows for avoiding data that could potentially cause harm to the system. This is important in allowing for the system to run smoothly and not be potentially hacked.  *Keep It Simple*: Keeping everything simple keeps a system from becoming too complicated and more difficult to protect.  *Adopt a Secure Coding Standard*: Having a secure standard is important for allowing the programmers to know what to do and what could be problematic. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astée | 22.10 | **“uninitialized-read”** (*EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This is labelled as a partial check. |
| Clang | 3.9 | “-Wuninitialized  clang-analyzer-core.UndefinedBinaryOperatorResult” (*EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This particular tool does not appear to catch every instance for this. Such examples are “uninitialized values read from heap-allocated memory”. (*EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| CodeSonar | 8.1p0 | “LANG.STRUCT.RPL  LANG.MEM.UVAR” (*EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This has “Return pointer to local Uninitialized variable” (*EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| Helix QAC | 2024.3 | “**DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978” (***EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| Klocwork | 2024.3 | **“UNINIT.CTOR.MIGHT UNINIT.CTOR.MUST UNINIT.HEAP.MIGHT UNINIT.HEAP.MUST UNINIT.STACK.ARRAY.MIGHT UNINIT.STACK.ARRAY.MUST UNINIT.STACK.ARRAY.PARTIAL.MUST UNINIT.STACK.MIGHT UNINIT.STACK.MUST” (***EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| LDRA Tool Suite | 9.7.1 | “53 D, 69 D, 631 S, 652 S” **(***EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This shows to have a parital implementation. |
| Parasoft C/C++test | 2023.1 | “CERT\_CPP-EXP53-a” **(***EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This says “avoid use before initialization”. (*EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| Parasoft Insure++ | NA | NA | This has a detection in the runtime. |
| Polyspace Bug Finder | R2024a | “CERT C++: EXP53-CPP” **(***EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This “Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered” (*EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.). |
| PVS-Studio | 7.33 | “V546, V573, V614, V670, V679, V730, V788, V1007, V1050” **(***EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| RuleChecker | 22.10 | **“uninitialized-read”** (*EXP53-CPP. Do Not Read Uninitialized Memory - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This is labelled as a partial check. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do not Attempt to Create a std::string from a Null Pointer** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | “The std::basic\_string type uses the traits design pattern to handle implementation details of the various string types, resulting in a series of string-like classes with a common, underlying implementation.” (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.)  This is a container that uses characters and uses the standard library. There are additional components that allow this to find null-terminated strings when using certain criteria and can end up taking a null pointer and dereferencing it from these patterns. |

Source, code examples, and other sections all comes from:

*STR51-CPP. Do not attempt to create a std::string from a null pointer - SEI CERT C++ Coding Standard - Confluence*. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer

| **Noncompliant Code** |
| --- |
| An example of noncompliant code can be taken from the std::string object. This can be exacerbated by using std::getenv(). The use of this can cause failure due to the getenv returning a null pointer. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| The compliant code example checks to make sure that the std::getenv() is set to null. This is done before the object is even created. |
| #include <cstdlib>  #include <string>    void f() {    const char \*tmpPtrVal = std::getenv("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):**  *Heed Compiler Warnings*: paying attention to the warnings that the compiler shows allows developers to find bugs, deprecated components, and other issues that could potentially open holes in the security. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | “assert\_failure” (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| CodeSonar | 8.1p0 | “LANG.MEM.NPD” (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This has “Null Pointer Dereference” (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| Helix QAC | 2024.3 | “DF4770, DF4771, DF4772, DF4773, DF4774” (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| Klocwork | 2024.3 | |  |  | | --- | --- | |  | **“NPD.CHECK.CALL.MIGHT NPD.CHECK.CALL.MUST NPD.CHECK.MIGHT NPD.CHECK.MUST NPD.CONST.CALL NPD.CONST.DEREF NPD.FUNC.CALL.MIGHT NPD.FUNC.CALL.MUST NPD.FUNC.MIGHT NPD.FUNC.MUST NPD.GEN.CALL.MIGHT NPD.GEN.CALL.MUST NPD.GEN.MIGHT NPD.GEN.MUST RNPD.CALL RNPD.DEREF”** (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.) | | No information given for this tool. |
| Parasoft C/C++test | 2023.1 | “CERT\_CPP-STR51-a” (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This one is labelled for “avoid null pointer dereferencing” (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| Polyspace Bug Finder | R2024a | “CERT C++: STR51-CPP” (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.) | “Checks for string operations on null pointer (rule partially covered).” (*STR51-CPP. Do Not Attempt to Create a Std::String From a Null Pointer - SEI CERT C++ Coding Standard - Confluence*, n.d.) |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | SQL Injections are vulnerabilities. This involves a SQL query that can be from untrusted data. This data is capable of altering the query maliciously and end up leaking information or changing the data. The best way to prevent this is to sanitize the data.  We can “suppose a system authenticates users by issuing the following query to a SQL database. If the query returns any results, authentication succeeds; otherwise, authentication fails.” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) |

Source, code examples, and other sections all comes from:

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

| **Noncompliant Code** |
| --- |
| The noncompliant example of this involves authenticating a user and putting their password through an array and a database connection. From there it’s possible to have the system hash out the password. |
| 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 compliant code example contains the use of parametric query and the character of ?. This allows for being a placeholder for an argument. Additionally, the code performs validation on the username to help prevent attacks when anything is submitted for longer than they should be. |
| 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 Data*: Since data comes from everywhere, SQL injections can take advantage of this. By validating the data that is being input into the system, it helps to prevent such attacks because the data is being checked first.  *Sanitize Data Sent to Other Systems*: Sanitizing the data that comes from an outside source is important for keeping attacks from slipping through the system. This helps prevent SQL injections because the problematic portions are removed.  *Adopt a Secure Coding Standard*: Adopting a secure standard allows for minimizing the access that such attacks can use. The development is being paid attention to and allowing for holes to be found and closed before they become a problem. |
| --- |

**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” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) | This tool labels for “Trust and security errors” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) |
| CodeSonar | 8.1p0 | “JAVA.IO.INJ.SQL” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) | Only lists for being an SQL injection. |
| Coverity | 7.5 | “SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) | Only shows for implementation. |
| Findbugs | 1.0 | “SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) | Only shows for implementation. |
| Fortify | 1.0 | “HTTP\_Response\_Splitting  SQL\_Injection\_\_Persistence  SQL\_Injection” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) | Only shows for implementation. |
| Klocwork | 2024.3 | “SV.DATA.DB  SV.SQL  SV.SQL.DBSOURCE” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) | Only shows for implementation. |
| Parasoft Jtest | 2024.1 | “CERT.IDS00.TDSQL” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) | Has a label of being able to protect against SQL injection attacks. |
| SonarQube | 9.9 | “S2077  S3649” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) | Lists 2 other rules:  “Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) |
| SpotBugs | 4.6.0 | “SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE  SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING” (*IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, n.d.) | Only shows for implementation. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Free Dynamically Allocated Memory when No Longer Needed** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Memory is allocated during the running of a program. When this memory is no longer needed, it must be released from the system in order to be used again.  “Before the lifetime of the last pointer that stores the return value of a call to a standard memory allocation function has ended, it must be matched by a call to free() with that pointer value.” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) |

Source, code examples, and other sections all comes from:

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

| **Noncompliant Code** |
| --- |
| The noncompliant code example shows that the object has been allocated, but not released when it is no longer needed. |
| #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** |
| --- |
| The compliant code shows that the pointer has been released when it is no longer needed through the use of the free() function. |
| #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):**  *Keep it Simple*: keeping the code simple is helpful in ensuring that the memory is freed when the system no longer needs it.  *Use Effective Quality Assurance Techniques*: when running tests and quality assurance, you need to ensure that everything works to a specific standard. By keeping the memory free, the tests can run smoothly, but also output the level of quality needed. Implementing the techniques will ensure that the quality continues.    *Adopt a Secure Coding Standard*: Having memory not be freed is a good way to cause security errors to appear in the code. By adopting a secure standard, it allows the program to remain secure and help minimize the possibility of not leaving such errors in. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | NA | Shows that there is no explicit checker, but is still supported. |
| Axivion Bauhaus Suite | 7.2.0 | “CertC-MEM31” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | This is capable of detecting the allocated resources that have not been freed. |
| CodeSonar | 8.1p0 | “ALLOC.LEAK” | This only shows a leak. |
| Compass/ROSE | NA | NA | |  | | --- | | No information given for this tool. | |
| Coverity | 2017.07 | “RESOURCE\_LEAK  ALLOC\_FREE\_MISMATCH” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | This is capable of finding resource leaks. |
| Cppcheck | 2.15 | “memleak  leakReturnValNotUsed  leakUnsafeArgAlloc  memleakOnRealloc” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | This does not have a return value for memory allocation. |
| Cppcheck Premium | 24.9.0 | “memleak  leakReturnValNotUsed  leakUnsafeArgAlloc  memleakOnRealloc” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | This does not have a return value for memory allocation. |
| Helix QAC | 2024.3 | “DF2706, DF2707, DF2708  C++3337, C++3338” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| Klocwork | 2024.3 | “CL.FFM.ASSIGN  CL.FFM.COPY  CL.SHALLOW.ASSIGN  CL.SHALLOW.COPY  FMM.MIGHT  FMM.MUST” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| LDRA tool suite | 9.7.1 | “50 D” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows that it has a partial implementation. |
| Parasoft C/C++test | 2023.1 | “CERT\_C-MEM31-a” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows that it makes sure that the resource is freed. |
| Parasoft Insure++ | NA | NA | This has a runtime analysis. |
| PC-lint Plus | 1.4 | “429” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | This has itself fully supported for freeing memory. |
| Polyspace Bug Finder | R2024a | “CERT C: Rule MEM31-C” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows that it has full coverage for memory leaks. |
| PVS-Studio | 47.33 | “V773” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| SonarQube C/C++ Plugin | 3.11 | “S3584” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| Splint | 3.11 | NA | No information given for this tool. |
| TrustInSoft Analyzer | 1.38 | “malloc” (*MEM31-C. Free Dynamically Allocated Memory When No Longer Needed - SEI CERT C Coding Standard - Confluence*, n.d.) | This tool has been verified. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a Static Assertion to Test the Value of a Constant Expression** |
| --- | --- | --- |
| **Assertions** | [STD-006-CCP] | Assertions are a diagnostic tool. This tool is helpful in finding vulnerabilities within the code and even look for leaks and overhead depending on what sub-function is being called. This tool does need to be placed within a proper function in order to be used. |

Source, code examples, and other sections all comes from:

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

| **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.” (*DCL03-C. Use a Static Assertion to Test the Value of a Constant Expression - SEI CERT C Coding Standard - Confluence*, n.d.) |
| #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 may be used, as in this compliant solution.” (*DCL03-C. Use a Static Assertion to Test the Value of a Constant Expression - SEI CERT C Coding Standard - Confluence*, n.d.) |
| 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*: the use of the compiler is to allow the programmers to know when there are issues and when using this, you’re able to have a view of what is happening and a basis to start from without spending a lot of time looking into the warnings.  *Adopt a Secure Coding Standard*: This allows for ensuring that the code is easily fixable and secure with less holes or fix the holes quickly. |
| --- |

**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” (*DCL03-C. Use a Static Assertion to Test the Value of a Constant Expression - SEI CERT C Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| Clang | 3.9 | “misc-static-assert” (*DCL03-C. Use a Static Assertion to Test the Value of a Constant Expression - SEI CERT C Coding Standard - Confluence*, n.d.) | This uses the clang-tidy to check. |
| CodeSonar | 8.1p0 | This is labelled as being customized. | This allows the people to “implement a custom check that reports uses of assert() macro”. (*DCL03-C. Use a Static Assertion to Test the Value of a Constant Expression - SEI CERT C Coding Standard - Confluence*, n.d.) |
| Compass/ROSE | NA | NA | This shows it can detect violations and see if it can evaluate it. |
| ÉCLAIR | 1.2 | “CC2.DCL03” (*DCL03-C. Use a Static Assertion to Test the Value of a Constant Expression - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows a full implementation. |
| LDRA tool suite | 9.7.1 | “44 S” (*DCL03-C. Use a Static Assertion to Test the Value of a Constant Expression - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows a full implementation. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Do Not Abruptly Terminate the Program** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | There are multiple functions for terminating a program, std::abort(), std::quick\_exit(), and std::\_Exit() are often used.  “They do so without calling exit handlers registered with std::atexit() and without executing destructors for objects with automatic, thread, or static storage duration. How a system manages open streams when a program ends is [implementation-defined](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-implementation-definedbehavior) [[ISO/IEC 9899:1999](https://wiki.sei.cmu.edu/confluence/display/cplusplus/AA.+Bibliography#AA.Bibliography-ISO-IEC9899-1999)]. Open streams with unwritten buffered data may or may not be flushed, open streams may or may not be closed, and temporary files may or may not be removed. Because these functions can leave external resources, such as files and network communications, in an indeterminate state, they should be called explicitly only in direct response to a critical error in the application. (See ERR50-CPP-EX1 for more information.)” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) |

Source, code examples, and other sections all comes from:

*ERR50-CPP. Do not abruptly terminate the program - SEI CERT C++ Coding Standard - Confluence*. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR50-CPP.+Do+not+abruptly+terminate+the+program

| **Noncompliant Code** |
| --- |
| In the noncompliant example, the main function is using std::atexit() to leave the program. The use of this can result in an std::terminate() result because there is the possibility of an exception occurring. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    int main() {    if (0 != std::atexit(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| The compliant example doesn’t use atexit by itself. It helps mitigate exceptions occurring through the use of the try and catch functions. These allow for the system to not terminate the program prematurely. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.    try {      throwing\_func();    } catch (...) {      // Handle error    }  }    int main() {    if (0 != std::atexit(f)) {      // Handle error    }    // ...  } |

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

| **Principles(s):**  *Use Effective Quality Assurance Techniques*: this is useful because, when testing, the system needs to have a level of quality that ensures a program runs as it should. By using the techniques, it allows for being aware of what can happen and handle it.  *Adopt a Secure Coding Standard*: as it stands, this is a continued use because it’s a way to keep people who have ill-intent from causing issues by terminating a program. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | “stdlib-use” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This shows as partially checked. |
| CodeSonar | 8.1p0 | “BADFUNC.ABORT  BADFUNC.EXIT” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This has use of abort and exit. |
| Helix QAC | 2024.3 | “C++5014” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| Klocwork | 2024.3 | “MISRA.TERMINATE  CERT.ERR.ABRUPT\_TERM” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| LDRA tool Suite | 9.7.1 | “122 S” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This indicates having ehanced enforcement. |
| Parasoft C/C++test | 2023.1 | |  |  | | --- | --- | |  | **“ CERT\_CPP-ERR50-a** **CERT\_CPP-ERR50-b** **CERT\_CPP-ERR50-c** **CERT\_CPP-ERR50-d** **CERT\_CPP-ERR50-e** **CERT\_CPP-ERR50-f** **CERT\_CPP-ERR50-g** **CERT\_CPP-ERR50-h** **CERT\_CPP-ERR50-i** **CERT\_CPP-ERR50-j** **CERT\_CPP-ERR50-k** **CERT\_CPP-ERR50-l** **CERT\_CPP-ERR50-m** **CERT\_CPP-ERR50-n”** (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | | “The execution of a function registered with 'std::atexit()' or 'std::at\_quick\_exit()' should not exit via an exception Never allow an exception to be thrown from a destructor, deallocation, and swap Do not throw from within destructor There should be at least one exception handler to catch all otherwise unhandled exceptions An empty throw (throw;) shall only be used in the compound-statement of a catch handler Exceptions shall be raised only after start-up and before termination of the program Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) Function called in global or namespace scope shall not throw unhandled exceptions Always catch exceptions Properly define exit handlers The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used Avoid throwing exceptions from functions that are declared not to throw The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| Polyspace Bug Finder | R2024a | “CERT C++: ERR50-CPP” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This checks for terminate function. |
| PVS-Studio | 7.33 | “V667, V2014” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| RuleChecker | 22.10 | “stdlib-use” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | Labelled as a partial check. |
| SonarQube C/C++ Plugin | 4.10 | “S990” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Detect and Handle Standard Library Errors** |
| --- | --- | --- |
| Error Handling | [STD-008-CPP] | Checking for errors is a good way to ensure that the data placed within the code comes back both correct and successful. To that end, keeping abreast of these errors allows for ensuring that the issues are handled, but also helps to reduce possible vulnerabilities in the code.  “Assuming that all calls to such functions will succeed and failing to check the return value for an indication of an error is a dangerous practice that may lead to [unexpected](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-unexpectedbehavior) or [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior) when an error occurs. It is essential that programs detect and appropriately handle all errors in accordance with an error-handling policy.” (*ERR33-C. Detect and Handle Standard Library Errors - SEI CERT C Coding Standard - Confluence*, n.d.) |

Source, code examples, and other sections all comes from:

*ERR33-C. Detect and handle standard library errors - SEI CERT C Coding Standard - Confluence*. (n.d.). https://wiki.sei.cmu.edu/confluence/display/c/ERR33-C.+Detect+and+handle+standard+library+errors

| **Noncompliant Code** |
| --- |
| The noncompliant example is focused around the setlocale function. This converts a UTF-8 character sequence. This code goes to set to the global locale, but has no check for any errors.  “The setlocale() function will fail by returning a null pointer, for example, when the locale is not installed. The function may fail for other reasons as well, such as the lack of resources.  Depending on the sequence of characters pointed to by utf8, the subsequent call to mbstowcs() may fail or result in the function storing an unexpected sequence of wide characters in the supplied buffer wcs.” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| #include <locale.h>  #include <stdlib.h>    int utf8\_to\_wcs(wchar\_t \*wcs, size\_t n, const char \*utf8,                  size\_t \*size) {    if (NULL == size) {      return -1;    }    setlocale(LC\_CTYPE, "en\_US.UTF-8");    \*size = mbstowcs(wcs, utf8, n);    return 0;  } |

| **Compliant Code** |
| --- |
| The compliant code example does check the setlocale() function to avoid anything that could make it fail. The idea behind this is that it can go back to initial settings when needed. |
| #include <locale.h>  #include <stdlib.h>    int utf8\_to\_wcs(wchar\_t \*wcs, size\_t n, const char \*utf8,                  size\_t \*size) {    if (NULL == size) {      return -1;    }    const char \*save = setlocale(LC\_CTYPE, "en\_US.UTF-8");    if (NULL == save) {      return -1;    }      \*size = mbstowcs(wcs, utf8, n);    if (NULL == setlocale(LC\_CTYPE, save)) {      return -1;    }    return 0;  } |

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

| **Principles(s):**  *Heed Compiler Warnings*: this principle is important in ensuring that the code contains few to no errors or warnings that could open holes.  *Adopt a Secure Coding Standard*: with an eye on warnings and errors, it is possible to remain more secure because the programmers are already watching for anything that could be problematic. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | “error-information-unused  Error-information-unused-computed” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | Listed as being partially checked. |
| Axivion Bauhaus Suite | 7.2.0 | “CertC-ERR33” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| CodeSonar | 8.1p0 | “LANG.FUNCS.IRV  LANG.ERRCODE.NOTEST  LANG.ERRCODE.NZ” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This shows as ignoring return values, missing a test, and has a non-zero code for errors. |
| Compass/ROSE | NA | NA | This is capable of detecting violations. |
| Coverity | 2017.07 | “MISRA C 2012 Rule 22.8  MISRA C 2012 Rule 22.9  MISRA C 2012 Rule 22.10” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This shows implementation. |
| Cppcheck Premium | 24.9.0 | “premium-cert-err33-c” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This shows a partial implementation. |
| Helix QAC | 2024.3 | “C3200  C++3802, C++3803, C++3804  DF2820, DF2821, DF2822, DF2823, DF2824, DF2930, DF2931, DF2932, DF2933, DF2934” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| Klocwork | 2024.3 | “NPD.CHECK.MUST  NPD.FUNC.MUST  SV.RVT.RETVAL\_NOTTESTED” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| LDRA tool suite | 9.7.1 | “80 D” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This shows a partial implementation. |
| Parasoft C/C++test | 2023.1 | “CERT\_C-ERR33-a  CERT\_C-ERR33-b  CERT\_C-ERR33-d” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | “The value returned by a standard library function that may return an error should be used The standard library functions for which return values need not be checked should be cast to 'void' Always check the returned value of non-void function” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| Parasoft Insure++ | NA | NA | Shows a runtime analysis. |
| PC-lint Plus | 1.4 | “534” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This shows as having partial support. |
| Polyspace Bug Finder | R2024a | “CERT C: Rule ERR33-C” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | “Checks for:   * Errno not checked * Return value of a sensitive function not checked * Unprotected dynamic memory allocation   Rule partially covered.” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) |
| RuleChecker | 24.04 | “error-information-unused” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This shows as having a partial check. |
| TrustInSoft Analyzer | 1.38 | “pointer arithmetic” (*ERR50-CPP. Do Not Abruptly Terminate the Program - SEI CERT C++ Coding Standard - Confluence*, n.d.) | This shows that it has been verified. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do Not Access Shared Objects in Signal Handlers** |
| --- | --- | --- |
| Signals | [STD-009-CPP] | One of the key components to programming is having consistent data. That allows for error handling, keeping vulnerabilities minimal, and an ease of searching when looking through large code sources. By not accessing the shared objects, it can keep this data consistent. This is important when a variable is a pointer and not a volatile variable.  “The signal handler may also call a handful of functions, including abort(). (See [SIG30-C. Call only asynchronous-safe functions within signal handlers](https://wiki.sei.cmu.edu/confluence/display/c/SIG30-C.+Call+only+asynchronous-safe+functions+within+signal+handlers) for more information.)” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) |

Source, code examples, and other sections all comes from:

*SIG31-C. Do not access shared objects in signal handlers - SEI CERT C Coding Standard - Confluence*. (n.d.). https://wiki.sei.cmu.edu/confluence/display/c/SIG31-C.+Do+not+access+shared+objects+in+signal+handlers

| **Noncompliant Code** |
| --- |
| The noncompliant example updates a character variable when a signal is delivered. The variable being called is a pointer variable and it is not volatile. |
| #include <signal.h>  #include <stdlib.h>  #include <string.h>    enum { MAX\_MSG\_SIZE = 24 };  char \*err\_msg;    void handler(int signum) {    strcpy(err\_msg, "SIGINT encountered.");  }    int main(void) {    signal(SIGINT, handler);      err\_msg = (char \*)malloc(MAX\_MSG\_SIZE);    if (err\_msg == NULL) {      /\* Handle error \*/    }    strcpy(err\_msg, "No errors yet.");    /\* Main code loop \*/    return 0;  } |

| **Compliant Code** |
| --- |
| The compliant code uses volatile sig\_atomic\_t and lets the handler only react on a variable with the type of sig\_atomic\_t. This set up also allows for more portability. |
| #include <signal.h>  #include <stdlib.h>  #include <string.h>    enum { MAX\_MSG\_SIZE = 24 };  volatile sig\_atomic\_t e\_flag = 0;    void handler(int signum) {    e\_flag = 1;  }    int main(void) {    char \*err\_msg = (char \*)malloc(MAX\_MSG\_SIZE);    if (err\_msg == NULL) {      /\* Handle error \*/    }      signal(SIGINT, handler);    strcpy(err\_msg, "No errors yet.");    /\* Main code loop \*/    if (e\_flag) {      strcpy(err\_msg, "SIGINT received.");    }    return 0;  } |

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

| **Principles(s):**  *Default Deny*: we don’t want to have the system to have access, especially for shared objects. Having a default deny system set up allows for avoiding issues that can come from such a problem.  *Adhere to the Principle of Least Privilege*: when you cannot have something denied, only allowing access to valid people or systems keeps from having everyone able to mess around.  *Practice Defense in Depth*: This is important because shared resources can be on many levels. Having the defense throughout the different layers helps to keep everything accessible and blocked depending on the level of the defense is rests in. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | “signal-handler-shared-access” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows as partially checked. |
| Axivion Bauhaus Suite | 7.2.0 | “CertC-SIG31” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| CodeSonar | 8.1p0 | “CONCURRENCY.DATARACE” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | Lists data race. |
| Compass/ROSE | NA | NA | This is capable of detecting violations in a single-file program. |
| Cppcheck Premium | 24.9.0 | “premium-cert-sig31-c” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows as a partial implementation. |
| Helix QAC | 2024.3 | “C2029, C2030, C++3854, C++3855” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| LDRA tool suite | 9.7.1 | “87 D” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows as having full implementation. |
| Parasoft C/C++test | 2023.1 | “CERT\_C-SIG31-a” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | This has signal handlers that are properly defined. |
| PC-lint Plus | 1.4 | “2765” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows full support. |
| Poyspace Bug Finder | R2024a | “CERT C: Rule SIG31-C” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | This checks the access that shared data has. |
| RuleChecker | 24.04 | “signal-handler-shared-access” (*SIG31-C. Do Not Access Shared Objects in Signal Handlers - SEI CERT C Coding Standard - Confluence*, n.d.) | This shows as being partially checked. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Never Hard Code Sensitive Information** |
| --- | --- | --- |
| Miscellaneous | STD-010-CPP | Sensitive information includes passwords and encryption keys. Hard coding these into a program leaves vulnerabilities open and exposing the sensitive information. If a hacker gets access to the code, this allows them the ability to see this information and exploit it. |

Source, code examples, and other sections all comes from:

*MSC41-C. Never hard code sensitive information - SEI CERT C Coding Standard - Confluence*. (n.d.). https://wiki.sei.cmu.edu/confluence/display/c/MSC41-C.+Never+hard+code+sensitive+information

| **Noncompliant Code** |
| --- |
| In the noncompliant example, the password is hardcoded into the code. This is done through a string that can be easily read in order to access something remotely. This exposes the password to everyone. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {    if (!authenticate("correct code")) {      printf("Authentication error\n");      return -1;    }      printf("Authentication successful\n");    // ...Work with system...    return 0;  } |

| **Compliant Code** |
| --- |
| The compliant example asks for the user to enter their authentication. This means that the system remains more secure because the code doesn’t just have the information. Only those with the information can get in instead of everyone. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  #define CODE\_LEN 50    char code[CODE\_LEN];    printf("Please enter your authentication code:\n");    fgets(code, sizeof(code), stdin);    int flag = authenticate(code);    memset\_s(code, sizeof(code), 0, sizeof(code));    if (!flag) {      printf("Access denied\n");      return -1;    }    printf("Access granted\n");    // ...Work with system...    return 0;  } |

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

| **Principles(s):**  *Architect and Design for Security Policies*: by including the security in the design, it is more likely that policies will be adhered to. By adhering to these things, developers are aware of what they can and cannot do, such as keeping sensitive information from being hard coded.  *Practice Defense in Depth*: when having multiple layers to security, hard coding can seem like a good idea. By practicing this, the developers and other uses do not have the sensitive information lying around.  *Adopt a Secure Coding Standard*: having a coding standard that remains secure allows for keeping developers aware of what they can and cannot do. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | NA | This shows as being supported. |
| CodeSonar | 8.1p0 | “HARDCODED.AUTH  HARDCODED.DNS  HARDCODED.KEY  HARDCODED.SALT  HARDCODED.SEED” (*MSC41-C. Never Hard Code Sensitive Information - SEI CERT C Coding Standard - Confluence*, n.d.) | This contains the different hard coding features for authentication, DNS name, crypto key, crypt sold, and PRNG. |
| Helix QAC | 2024.3 | “C3122  C++3842” (*MSC41-C. Never Hard Code Sensitive Information - SEI CERT C Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| Klocwork | 2024.3 | “HCC  HCC.PWD  HCC.USER  CXX.SV.PWD.PLAIN  CXX.SV.PWD.PLAIN.LENGTH  CXX.SV.PWD.PLAIN.ZERO” (*MSC41-C. Never Hard Code Sensitive Information - SEI CERT C Coding Standard - Confluence*, n.d.) | No information given for this tool. |
| Parasoft C/C++test | 2023.1 | “CERT\_C-MSC41-a” (*MSC41-C. Never Hard Code Sensitive Information - SEI CERT C Coding Standard - Confluence*, n.d.) | This does not allow code literal strings. |
| PC-lint Plus | 1.4 | “2460” (*MSC41-C. Never Hard Code Sensitive Information - SEI CERT C Coding Standard - Confluence*, n.d.) | “Assistance provided: reports when a literal is provided as an argument to a function parameter with the ‘noliteral’ argument Semantic; several Windows API functions are marked as such and the ‘-sem’ option can apply it to other functions as appropriate” (*MSC41-C. Never Hard Code Sensitive Information - SEI CERT C Coding Standard - Confluence*, n.d.) |
| Polyspace Bug Finder | R2024a | “CERT C: Rule MSC41-C” | Checks if there is sensitive data hard coded into the program. |
| RuleChecker | 24.014 | NA | This shows it has support. |

### 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.

Many of the phases that exist in DevOps can include security. This is important in helping the process. Adding security in turns DevOps into DevSecOps. The Assess, Plan, and Design phases have these elements already shown through the use of the diagram. These phases allow for planning how security is a part of the process, designing it in so that it is automatically done, and programming it into the system as an important part of the product. It continues from there by including security in the verify and test phase through the use of static application testing and automated security scans. A lot of this can also be added into the unit, integration, and other testing features that are important for ensuring that the security of the system is not on the back burner.

When the development phase has been finished and shows the level of security needed, everything moves into the production phase. The use of security within this would be ensuring that the system has the security settings configured and has been tested to be as secure as possible. Many methods for maintaining security once in the realm of production flow through to the different areas. These can include keeping abreast of current vulnerability reports to help maintain the production with updates and new baselines. When an attack does happen, responding to keep the attack surface minimized, while patching any holes found is taken care of. This is all about monitoring and testing in real-time. This helps to keep the situation up-to-date for when information shows that there could potentially be a risk.

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

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest is a way to protect the data stored in the system. This can include the hard drives, phones, computers, cloud assets, and others items. This protection of the data can be done through the use of encryption tools, disk encryption, and the security of the devices. The use of this inside of the system allows for ensuring that the data is still safe, even when it is not moving. By protecting this data, the policy is being applied to keep the data secure when it gets to its destination. |
| Encryption in flight | Encryption in flight is for protecting data that is moving through the system. There are many ways that it can be used. This can be through the use of a robust network with firewalls and authentication, but also through areas that are outside of the network. This applies in that the data travels across devices and outside of the network. |
| Encryption in use | When data is being used, it still needs to have a level of protection. When using encryption for in use data, there is limiting the vulnerability of when data is decrypted to be used normally. The use of this feature as a part of the policy allows the data to remain secure and removes the change of the data being placed into a plaintext format. |

Sources:

*Data Protection: Data In transit vs. Data At Rest*. (n.d.). Fortra’s Digital Guardian. https://www.digitalguardian.com/blog/data-protection-data-in-transit-vs-data-at-rest

*IBM Aspera on cloud*. (n.d.). https://www.ibm.com/docs/en/aspera-on-cloud?topic=encryption-content-in-flight-rest

Velimirovic, A. (2023, November 16). *Data encryption in use explained*. phoenixNAP Blog. https://phoenixnap.com/blog/encryption-in-use#:~:text=Encryption%20in%20use%20is%20the,without%20the%20need%20for%20decryption.

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication helps in being “the process of identifying a user and granting them access to the network” (Martinez, 2024). This is commonly through the use of username and passwords. This policy applies in that it is a way to allow for ensuring someone is who they say they are. An additional component is adding multi-factor authentication, such as sending a code to a device. Using this helps to protect the data by keeping unknown individuals from accessing the system. |
| Authorization | When authorizing a user, the system “enforces the network policies, granular access control, and user privileges” (Martinez, 2024). Users need to be regulated to ensure they don’t access data and areas of the network that they don’t need to. This applies to the security policy of least privilege. Applying this policy keeps with proper security principles. |
| Accounting | When adding accounting into the triple-A framework, the idea is that it’s keep track of what is in the network. This is where “the protocol… collect[s] and log[s] data on user sessions, such as length of time, type of session, and resource usage” (Martinez, 2024). This is an important policy as it allows for ensuring that the network is always maintained and the privileges of the users is correct. It is also good for deactivating old users that may have slipped through the cracks after they left. |

Martinez, J. (2024, September 27). What is AAA Security? Authentication, Authorization, and Accounting. *strondm*. Retrieved December 7, 2024, from https://www.strongdm.com/blog/aaa-security

**\***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 | 11/14/2024 | Module 3 Milestone | Samantha Durr | Samantha Durr |
| 1.2 | 12/7/2024 | Project 1 | Samantha Durr | Samantha Durr |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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