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



# Green Pace Secure Development Policy

## Contents

Acknowledgements 2

[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)

## Acknowledgements

All coding examples in the coding blocks of this paper as well as explanations in some explanation blocks were extracted from the SEI CERT website at: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/2+Rules>

## 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 | Validating input data ensures that only properly formed data is being accepted into the system. Data that is not properly formed can cause malfunctions, bugs attacks and vulnerabilities. Each system is unique to the form in which data is accepted by that system therefore, failed validations should be considered risks. |
| 1. Heed Compiler Warnings | Compiler warnings are an indication of something bigger, an issue with the code and should not be taken lightly. Syntax issues usually lead to compiler warnings which can lead to compiler errors and the program fails to read the machine code. When this happens, the system will fail and with systems down trojans can be planted and harm the system once it is back online. |
| 1. Architect and Design for Security Policies | Following OWASP’ secure architecture (SA) practices is recommended as well as the use of subsystems, reliable libraries and plugins. SEI CERT also provides a list of rules and policies that can be used as a reference for security policies. |
| 1. Keep It Simple | Using simple user friendly designs is the desired thing to do. Security controls should not be overwhelming and easy to navigate and understand. The code should be readable and in most cases reusable. |
| 1. Default Deny | By default, no one without authorization should be able to access the system. This policy should be applied across the board to new and old user accounts and access accounts. New settings and changes, features should all be secure until its time to release them. |
| 1. Adhere to the Principle of Least Privilege | Permissions should be set for all requests, changes, and access attempts and they must be validated. It should be rule of thumb to validate permissions before any new feature is released and permissions should be reviewed often. |
| 1. Sanitize Data Sent to Other Systems | String data passed to other systems may contain special characters that can trigger commands or actions, resulting in a software vulnerability. As a result, it is necessary to sanitize all string data passed to other systems so that the resulting string is innocuous in the context in which it will be interpreted. |
| 1. Practice Defense in Depth | Multiple layers of security stacked on top of each other in a single system. The idea is to have the security protocols of the systems checking each other for vulnerabilities working in an umbrella-type environment. |
| 1. Use Effective Quality Assurance Techniques | Feedback is always key with new developments and projects, so it is always advised to have QA involved from initial planning. Regular product testing and improvements made around proven tests and test results are encouraged for a better user experience with the product. |
| 1. Adopt a Secure Coding Standard | Security standards that promote secure coding have been put in place over the years. Best and recommended practices have been published and should be followed because they come from a place of deep research and development. Developers need to be aware of these secure coding standards and best practices, so it becomes second nature when they are developing code. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CLG] | **DCL12-C. Implement abstract data types using opaque types** - Abstract data types are not restricted to object-oriented languages such as C++ and Java. They should be created and used in C language programs as well (Sei Cert, 2016). |

| **Noncompliant Code** |
| --- |
| The managed string type and the functions that operate on this type are defined in the string\_m.h header file. This noncompliant code example is based on the managed string library developed by CERT [[Burch 2006](https://wiki.sei.cmu.edu/confluence/display/c/AA.+Bibliography#AA.Bibliography-Burch06)]. |
| 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** |
| --- |
| This compliant solution reimplements the string\_mx type as a private type, hiding the implementation of the data type from the user of the managed string library (Sei Cert, 2016). |
| 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);  /\* … \*/ |
| In the internal header file, struct string\_mx is fully defined but not visible to a user of the data abstraction: |
| struct string\_mx {  size\_t size;  size\_t maxsize;  unsigned char strtype;  char \*cstr;  }; |

| **Principles(s):** Validate Input Data, Adopt a Secure Coding Standard, Architect and Design for Security Policies - Abstract data types are not restricted to object-oriented languages such as C++ and Java. Security standards like this promote secure coding and have been put in place over the years. Following OWASP’ secure architecture (SA) practices is recommended as well as the use of subsystems, reliable libraries and plugins. |
| --- |

**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-DCL12** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **104 D** | Partially implemented |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rec. DCL12-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.dcl12c.html) | Checks for structure or union object implementation visible in file where pointer to this object is not dereferenced (rule partially covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 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** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CLG] | **INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data –** According to the Sei Cert database, Integer conversions, both implicit and explicit (using a cast), must be guaranteed not to result in lost or misinterpreted data (2016). |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type (Sei Cert, 2016). |
| #include <limits.h>    void func(void) {     unsigned long int u\_a = ULONG\_MAX;     signed char sc;     sc = (signed char)u\_a; /\* Cast eliminates warning \*/     /\* … \*/  } |

| **Compliant Code** |
| --- |
| Validate ranges when converting from an unsigned type to a signed type. |
| #include <limits.h>    void func(void) {    unsigned long int u\_a = ULONG\_MAX;    signed char sc;    if (u\_a <= SCHAR\_MAX) {      sc = (signed char)u\_a;  /\* Cast eliminates warning \*/    } else {      /\* Handle error \*/    }  } |

| **Principles(s):** ValidateInput Data, Architect and Design for Security Policies - Each system is unique to the form in which data is accepted by that system therefore, failed validations should be considered risks. SEI CERT provides a list of rules and policies that can be used as a reference for security policies architect and design. |
| --- |

**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) | 22.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.CAST.PC.AV LANG.CAST.PC.CONST2PTR LANG.CAST.PC.INT**  **LANG.CAST.COERCE LANG.CAST.VALUE**  **ALLOC.SIZE.TRUNC MISC.MEM.SIZE.TRUNC**  **LANG.MEM.TBA** | Cast: arithmetic type/void pointer Conversion: integer constant to pointer Conversion: pointer/integer  Coercion alters value Cast alters value.  Truncation of allocation size Truncation of size  Tainted buffer access |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule. However, false warnings may be raised if limits.h is included |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **NEGATIVE\_RETURNS**  **REVERSE\_NEGATIVE**  **MISRA\_CAST** | Can find array accesses, loop bounds, and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior.  Can find instances where a negativity check occurs after the negative value has been used for something else.  Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CLG] | **STR30-C: Do not attempt to modify string literals.** Modifying a string literal frequently results in an access violation because string literals are typically stored in read-only memory (Sei Cert, 2016). |

| **Noncompliant Code** |
| --- |
| Char pointer str is initialized to the address of a string literal and changing the string literal can be unpredictable. |
| **Char** \*str  = “string literal”;  str[0] = ‘S’; |

| **Compliant Code** |
| --- |
| The string literal denotes the initial values of objects in the array hence determining its size. |
| **Char** str[] = “string literal”;  str[0] = ‘S’; |

| **Principles(s):** ValidateInput Data, Sanitize Data Sent to Other Systems, Heed Compiler Warnings - Validating input data ensures that only properly formed data is being accepted into the system and string correctness is vital. String data passed to other systems may contain special characters that can trigger commands or actions, resulting in a software vulnerability. Syntax issues (caused by string correctness) usually lead to compiler warnings which can lead to compiler errors and the program fails to read the machine code. |
| --- |

**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) | 22.04 | **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 \*” |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **157 S** | Partially implemented |

#### 

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAV] | **IDS00-J. Prevent SQL injection –** SQL query can have elements originating from untrusted sources causing SQL Injection weaknesses and attacks. |

| **Noncompliant Code** |
| --- |
| The code below  shows JDBC code to authenticate a user to a system. Password are passed as char and hashed but this code is still vulnerable to SQL Injection attacks. The unsanitized input argument “username” can be incorporated into the SQL command. |
| 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:hartoft: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** |
| --- |
| In the code below, a ? character is used as a placeholder for the argument. This is set up as a parametric query which validates the length of the username. This prevents attacks with usernames that are too long. |
| Public void doPrivilegedAction(    String username, char[] password  ) throws SQLException {    Connection connection = getConnection();    if (connection == null) {      // Handle error    }    try {      String pwd = hashPassword(password);        // Validate username length      if (username.length() > 8) {        // Handle error      }        String sqlString =        “select \* from db\_user where username=? And password=?”;      PreparedStatement stmt = connection.prepareStatement(sqlString);      stmt.setString(1, username);      stmt.setString(2, pwd);      ResultSet rs = stmt.executeQuery();      if (!rs.next()) {        throw new SecurityException(“User name or password incorrect”);      }        // Authenticated; proceed    } finally {      try {        connection.close();      } catch (SQLException x) {        // Forward to handler      }    }  } |

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

| **Principles(s):** ValidateInput Data, Default Deny, Practice Defense in Depth, Adhere to the Principle of Least Privilege, Adopt a Secure Coding Standard - Data that is not properly formed can cause malfunctions, bugs attacks and vulnerabilities. By default, no one without authorization should be able to access the system and this stops SQL Injection attacks. Defense in Depth is multiple layers of security stacked on top of each other in a single system working together to prevent threats like SQL Injection attacks. Permissions should be set for all requests, changes, and access attempts and they must be validated. Best and recommended practices have been published and should be followed because they come from a place of deep research and development. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L2 |

**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** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | **MEM50-CPP: Do not access freed memory** – Memory Protection is a very important standard that must be adhered to protect from know vulnerabilities like arbitrary code execution. |

| **Noncompliant Code** |
| --- |
| The none compliant code below shows how the rule MEM%)-CPP is not being followed. S is dereferenced after it has been deallocated, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process (Sei Cert, 2016). |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // …  delete s;  // …  s->f();  } |

| **Compliant Code** |
| --- |
| The dynamically allocated memory is not deallocated until it is no longer needed (Sei Cert, 2016). |
| #include <new>    struct S {     void f();  };    void g() noexcept(false) {     S \*s = new S;     // …     s->f();     delete s;  } |

| **Principles(s):** Sanitize Data Sent to Other Systems, Adopt a Secure Coding Standard, Architect and Design for Security Policies - It is necessary to sanitize all string data passed to other systems so that the resulting string is innocuous in the context in which it will be interpreted for memory protection purposes. Secure coding practices will also help with memory protection. SEI CERT provides a list of rules and policies that can be used as a reference for security policies. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy but does not catch all violations of this rule. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **ALLOC.UAF** | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | **USE\_AFTER\_FREE** | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | **DCL03-C. Use a static assertion to test the value of a constant expression –** Assertions are used to detect and eliminate program defects that may cause vulnerabilities in the software**.** |

| **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 |
| #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 (Sei Cert, 2016). |
| 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 |

| **Principles(s):** Sanitize Data Sent to Other Systems, Adopt a Secure Coding Standard – String data passed to other systems may contain special characters that can trigger commands or actions, resulting in a software vulnerability. Assertions can be used to prevent this from occurring. There are coding practices in place for implementing assertions and they should be followed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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 |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | **ERR51-CPP. Handle all exceptions.** When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block in which the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks of the same thread. |

| **Noncompliant Code** |
| --- |
| The example below shows neither f() nor main() catching the exceptions thrown by throwing\_func(), std::terminate() is called because no handler is present. |
| Void throwing\_func() noexcept(false);    void f() {     throwing\_func();  }    int main() {     f();  } |

| **Compliant Code** |
| --- |
| Below, the main entry point handles all exceptions, this ensures that the stack is unwound. |
| Void throwing\_func() noexcept(false);    void f() {     throwing\_func();  }    int main() {     try {       f();     } catch (…) {       // Handle error     }  } |

| **Principles(s):** ValidateInput Data, Heed Compiler Warnings, Default Deny - All exceptions thrown by an application must be caught by a matching exception handler. Even if the exception cannot be gracefully recovered from, using the matching exception handler ensures that the stack will be properly unwound and provides an opportunity to gracefully manage external resources before terminating the process. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions 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 |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Management** | [STD-008-CLG] | **MEM31-C. Free dynamically allocated memory when no longer needed -** 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. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the object allocated by the call to malloc() is not freed before the end of the lifetime of the last pointer text\_buffer referring to the object (Sei Cert, 2016). |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {    char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);    if (text\_buffer == NULL) {      return -1;    }    return 0;  } |

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

| **Principles(s):** Architect and Design for Security Policies, Adopt a Secure Coding Standard, Sanitize Data Sent to Other Systems, Adhere to the Principle of Least Privilege – A hardware device called a memory management unit (MMU) is usually used to convert virtual addresses into physical addresses stored on the MMU. Memory Management coding practices are used for best results with memory management of the code. **NOTE:** **Allocated memory does not need to be freed if it is assigned to a pointer whose lifetime includes program termination.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM31** | Can detect dynamically allocated resources that are not freed |
| [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 |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **leakReturnValNotUsed** | Doesn’t use return value of memory allocation function |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **50 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-MEM31-a** | Ensure resources are freed |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Characters and Strings (STR)** | [STD-009-CPP] | **STR53-CPP. Range check element access –** The std::string index operators const\_reference operator[ ](size\_type) const and reference operator[ ](size\_type) return the character stored at the specified position, pos. When pos >= size(), a reference to an object of type hart with value hart() is returned (Sei Cert, 2016). |

| **Noncompliant Code** |
| --- |
| The value returned by the call to get\_index() may be greater than the number of elements stored. |
| #include <string>    extern std::size\_t get\_index();    void f() {    std::string s(“01234567”);    s[get\_index()] = ‘1’;  } |

| **Compliant Code** |
| --- |
| The std::basic\_string::at() function is used in the code below and is set to throw an exception. |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();    void f() {    std::string s(“01234567”);    try {      s.at(get\_index()) = ‘1’;    } catch (std::out\_of\_range &) {      // Handle error    }  } |

| **Principles(s):** ValidateInput Data, Heed Compiler Warnings, Architect and Design for Security Policies, - Software programs often contain multiple components that act as subsystems wherein each component operates in one or more trusted domains. For example, one component may have access to the file system but lack access to the network, while another component has access to the network but lacks access to the file system (Sei Cert, 2016). The std::string index operators const\_reference operator[](size\_type) const and reference operator[](size\_type) return the character stored at the specified position, pos. When pos >= size(), a reference to an object of type charT with value charT() is returned. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.MEM.BO** **LANG.MEM.BU** **LANG.MEM.TBA** **LANG.MEM.TO** **LANG.MEM.TU** | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3162, C++3163, C++3164, C++3165** | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-010-CLG] | **EXP47-C. Do not call va\_arg with an argument of the incorrect type -** The variable arguments passed to a variadic function are accessed by calling the va\_arg() macro (Sei Cert, 2016). |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to read a variadic argument of type unsigned char with va\_arg(). |
| #include <stdarg.h>  #include <stddef.h>    void func(size\_t num\_vargs, …) {    va\_list ap;    va\_start(ap, num\_vargs);    if (num\_vargs > 0) {      unsigned char c = va\_arg(ap, unsigned char);      // …    }    va\_end(ap);  }    void f(void) {    unsigned char c = 0x12;    func(1, c);  } |

| **Compliant Code** |
| --- |
| The compliant solution accesses the variadic argument with type int, and then casts the resulting value to type unsigned char (Sei Cert, 2016) |
| #include <stdarg.h>  #include <stddef.h>    void func(size\_t num\_vargs, …) {    va\_list ap;    va\_start(ap, num\_vargs);    if (num\_vargs > 0) {      unsigned char c = (unsigned char) va\_arg(ap, int);      // …    }    va\_end(ap);  }    void f(void) {    unsigned char c = 0x12;    func(1, c);  } |

| **Principles(s):** Adopt a Secure Coding Standard, Sanitize Data Sent to Other Systems - Developers need to be aware of these secure coding standards and best practices, so it becomes second nature when they are developing code. String data passed to other systems may contain special characters that can trigger commands or actions, resulting in a software vulnerability. As a result, it is necessary to sanitize all string data passed to other systems so that the resulting string is innocuous in the context in which it will be interpreted. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wvarargs | Can detect some instances of this rule, such as promotable types. Cannot detect mismatched types or incorrect number of variadic arguments. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **BADMACRO.STDARG\_H** | Use of <stdarg.h> feature |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **44 S** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-EXP47-a** | Do not call va\_arg with an argument of the incorrect type |

### 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.

### 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-CLG | Low | Unlikely | High | P1 | 3 |
| STD-002-CLG | High | Probable | High | P6 | 2 |
| STD-003-CLG | Low | Likely | Low | P9 | 2 |
| STD-004-JAV | High | Probable | Medium | P12 | 2 |
| STD-005-CPP | High | Likely | Medium | P18 | 1 |
| STD-006-CLG | Low | Unlikely | High | P1 | 3 |
| STD-007-CPP | Low | Probable | Medium | P4 | 3 |
| STD-008-CLG | Medium | Probable | Medium | P8 | 2 |
| STD-009-CPP | High | Unlikely | Medium | P6 | 2 |
| STD-010-CLG | Medium | Likely | High | P6 | 2 |

### Create Policies for Encryption and Triple A

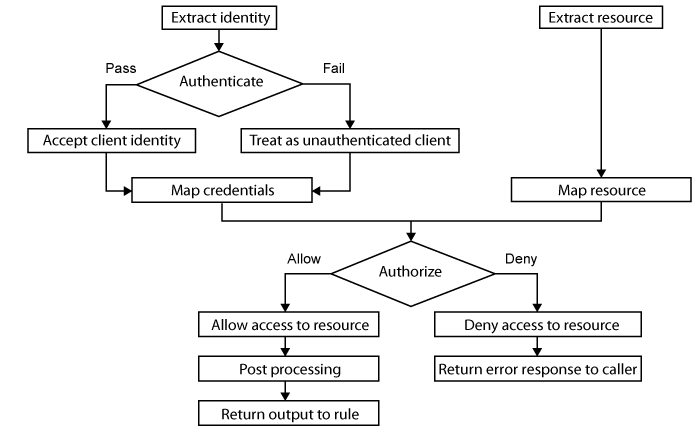
Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 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 | Data At Rest Encryption (DARE) is the encryption of the data that is stored in the databases and is not moving through networks. With DARE, data at rest including offline backups are protected. According to Microsoft, “Encryption at rest is designed to prevent the attacker from accessing the unencrypted data by ensuring the data is encrypted when on disk. If an attacker obtains a hard drive with encrypted data but not the encryption keys, the attacker must defeat the encryption to read the data” (MSM Baldwin, 2016). |
| Encryption at flight | Encryption at flight is the encryption of data that moves over a network. Basically, it is the encryption of data in transit. When data moves over a network it is vulnerable to interceptions over the network creating points of weakness. |
| Encryption in use | The general encryption methods only protect data when it is at rest (disk encryption), and when it is in transit of networks. This leaves vulnerabilities when the data is in use by on-premises or cloud applications. Encryption policies to implement protection of data in use are practiced by organizations protecting cloud information. This produces weak endpoints with cloud based systems and storage. According to Microsoft, “In-Use encryption takes a new approach that ensures that sensitive data is never left unsecured, regardless of or lifecycle stage (at rest, in transit, or in use) source, or location (on premise, cloud, or hybrid)” (MSM Baldwin, 2016). |

**TRIPLE A FRAMEWORK PROCESS FLOW CHART**



| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | After the AAA policy extracts the service requester identity and resource, it authenticates the claimed identity. Authentication is most accomplished through an external service (for example, a RADIUS or LDAP server), but custom processing methods, such as site-specific XML or XPath based solutions, are supported. During policy definition, you select a single authentication method, and, depending on the selected method, provide more required information. |
| Authorization | Successful server-based authentication generates a set of credentials that attest to the identity of the service requester. You can then map these credentials to a set that is more appropriate to the authorization method. You can accomplish this optional mapping through an XPath expression, an XML mapping file, or a custom method.  As with identity credentials, the extracted resource name can be mapped to a more appropriate authorization method. The methods to achieve this optional mapping are the same as the methods for credential mapping.  The resulting credentials, along with the resultant resource name, are the basis for client authorization. Client authorization determines whether the identified client has access to the requested resource. |
| Accounting | Accounting, the final process in the framework, is all about measuring what's happening within the network. As part of the protocol, it will collect and log data on user sessions, such as length of time, type of session, and resource usage. The value here is that it offers a clear audit trail for compliance and business purposes. |

**\***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.

|  |  |  |
| --- | --- | --- |
| **PRINCIPLE** | **STANDARD** | **JUSTIFICATION** |
| **1** | **001, 002, 003,** | **Principle 1 is about Validating Input Data. The 3 Standards are all about data inputting** |
| **2** | **003, 004, 006, 007, 009, 010** | **Compiler warnings are an indication of something bigger, an issue with the code and should not be taken lightly. Syntax issues usually lead to compiler warnings which can lead to compiler errors and the program fails to read the machine code. When this happens, the system will fail and with systems down trojans can be planted and harm the system once it is back online.** |
| **3** | **003, 004, 005, 006, 007, 008, 009, 010** | **Following OWASP’ secure architecture (SA) practices is recommended as well as the use of subsystems, reliable libraries and plugins. SEI CERT also provides a list of rules and policies that can be used as a reference for security policies.** |
| **4** | **001, 002, 003, 004, 005, 006, 007, 008, 009, 010** | **Security controls should not be overwhelming and easy to navigate and understand. The code should be readable and in most cases reusable.** |
| **5** | **001, 002, 003, 009,** | **This policy should be applied across the board to new and old user accounts and access accounts. New settings and changes, features should all be secure until it’s time to release them.** |
| **6** | **001, 003, 004, 007, 009, 010** | Permissions should be set for all requests, changes, and access attempts and they must be validated. It should be rule of thumb to validate permissions before any new feature is released and permissions should be reviewed often. |
| **7** | **003, 004, 005, 006, 007** | **String data passed to other systems may contain special characters that can trigger commands or actions, resulting in a software vulnerability.** |
| **8** | **004, 005, 006, 007, 008, 009** | **Multiple layers of security stacked on top of each other in a single system. The idea is to have the security protocols of the systems checking each other for vulnerabilities working in an umbrella-type environment.** |
| **9** | **001, 010** | **Feedback is always key with new developments and projects, so it is always advised to have QA involved from initial planning. Regular product testing and improvements made around proven tests and test results are encouraged for a better user experience with the product.** |
| **10** | **001, 002, 003, 004, 005, 006, 007, 008, 009, 010** | **Security standards that promote secure coding have been put in place over the years. Best and recommended practices have been published and should be followed because they come from a place of deep research and development. Developers need to be aware of these secure coding standards and best practices, so it becomes second nature when they are developing code.** |

**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 | 03/19/2023 | Coding Standards | Shingirai Chiremba | [Insert text.] |
| 1.2 | 04/6/2023 |  | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

**References**

MSM Baldwin. (2016). Azure Data Encryption-at-rest - azure security. Azure Data Encryption-at-Rest - Azure

Security | Microsoft Learn. Retrieved April 9, 2023, from <https://learn.microsoft.com/en-us/azure/security/fundamentals/encryption-atrest>

SEI CERT (2016), C Coding Standard: Rules for Developing Safe, Reliable, and Secure Systems in Software

Engineering Institute | Carnegie Mellon University [DISTRIBUTION STATEMENT A] Approved for public release and unlimited distribution.

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/2+Rules>