**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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 14](#_Toc52464063)

[Coding Standard 6 17](#_Toc52464064)

[Coding Standard 7 19](#_Toc52464065)

[Coding Standard 8 21](#_Toc52464066)

[Coding Standard 9 23](#_Toc52464067)

[Coding Standard 10 25](#_Toc52464068)

[Defense-in-Depth Illustration 27](#_Toc52464069)

[Project One 27](#_Toc52464070)

[1. Revise the C/C++ Standards 27](#_Toc52464071)

[2. Risk Assessment 27](#_Toc52464072)

[3. Automated Detection 27](#_Toc52464073)

[4. Automation 27](#_Toc52464074)

[5. Summary of Risk Assessments 28](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 28](#_Toc52464076)

[7. Map the Principles 29](#_Toc52464077)

[Audit Controls and Management 30](#_Toc52464078)

[Enforcement 30](#_Toc52464079)

[Exceptions Process 30](#_Toc52464080)

[Distribution 31](#_Toc52464081)

[Policy Change Control 31](#_Toc52464082)

[Policy Version History 31](#_Toc52464083)

[Appendix A Lookups 31](#_Toc52464084)

[Approved C/C++ Language Acronyms 31](#_Toc52464085)

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validate input from all untrusted data sources. Proper input validation can eliminate most software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user-controlled files [Seacord 05]. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code [C MSC00-A, C++ MSC00-A]. Use static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set. |
| 1. Keep It Simple | Keep the design as simple and small as possible [Saltzer 74, Saltzer 75]. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied, and the protection scheme identifies conditions under which access is permitted [Saltzer 74, Saltzer 75]. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker must execute arbitrary code with elevated privileges [Saltzer 74, Saltzer 75]. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems [C STR02-A] such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components using SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment [Seacord 05]. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions [Seacord 05]. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

### 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-PRE30-C] | Do not create a universal character name through concatenation |

| **Noncompliant Code** |
| --- |
| This code example is noncompliant because it produces a universal character name by token concatenation: |
| #define assign(uc1, uc2, val) uc1##uc2 = val    **void** func(**void**) {  **int** \u0401;    /\* ... \*/    assign(\u04, 01, 4);    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution uses a universal character name but does not create it by using token concatenation: |
| #define assign(ucn, val) ucn = val    **void** func(**void**) {  **int** \u0401;    /\* ... \*/    assign(\u0401, 4);    /\* ... \*/  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | **universal-character-name-concatenation** | Fully Checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-PRE30** | Fully Implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **LANG.PREPROC.PASTE LANG.PREPROC.PASTEHASH** | Macro uses ## operator ## follows # operator |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | **C0905**  **C++0064,C++0080** | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-INT30-C] | Ensure that unsigned integer operations do not wrap |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in an unsigned integer wrap during the addition of the unsigned operands ui\_and ui\_b. If this behavior is unexpected, the resulting value may be used to allocate insufficient memory for a subsequent operation or in some other manner that can lead to an exploitable vulnerability. |
| **void** func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum = ui\_a + ui\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution performs a precondition test of the operands of the addition to guarantee there is no possibility of unsigned wrap: |
| PRECONDITION TEST  #include <limits.h>    **void** func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum;  **if** (UINT\_MAX - ui\_a < ui\_b) {      /\* Handle error \*/    } **else** {      usum = ui\_a + ui\_b;    }    /\* ... \*/  }  POSTCONDITION TEST  **void** func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum = ui\_a + ui\_b;  **if** (usum < ui\_a) {      /\* Handle error \*/    }    /\* ... \*/  } |

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

| Addition is between two operands of arithmetic type or between a pointer to an object type and an integer type. This rule applies only to addition between two operands of arithmetic type.)  Incrementing is equivalent to adding 1. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | **universal-character-name-concatenation** | Fully Checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-INT30** | Implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | **C2910, C2911, C2912, C2913, C3383, C3384, C3385, C3386**  **C++2910, C++2911, C++2912, C++2913** |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib>  #include <string>  void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>  void f() {  const char \*tmpPtrVal = std ::getenv(“TMP”);  std::string tmp(tmpPtrVal ? tmpPrtVal : “ ”);  if (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** : Dereferencing a null pointer is undefined behavior, typically abnormal program termination. In some situations, however, dereferencing a null pointer can lead to the execution of arbitrary code [Jack 2007, van Sprundel 2006]. The indicated severity is for this more severe case; on platforms where it is not possible to exploit a null pointer dereference to execute arbitrary code, the actual severity is low. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert\_failure |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP\_STR5-a |  |
|  |  |  |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-IDS00-J-LLL] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows JDBC code to authenticate a user to a system. The password is passed as a char array, the database connection is created, and then the passwords are hashed.  Unfortunately, this code example permits a SQL injection attack by incorporating the unsanitized input argument username into the SQL command, allowing an attacker to inject validuser' OR '1'='1. The password argument cannot be used to attack this program because it is passed to the hashPassword() function, which also sanitizes the input. |
| **import** java.sql.Connection;  **import** java.sql.DriverManager;  **import** java.sql.ResultSet;  **import** java.sql.SQLException;  **import** java.sql.Statement;    **class** Login {  **public** Connection getConnection() **throws** SQLException {      DriverManager.registerDriver(**new**              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  **return** DriverManager.getConnection(dbConnection);    }      String hashPassword(**char**[] password) {      // Create hash of password    }    **public** **void** doPrivilegedAction(String username, **char**[] password)  **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**) {        // Handle error      }  **try** {        String pwd = hashPassword(password);          String sqlString = "SELECT \* FROM db\_user WHERE username = '"                           + username +                           "' AND password = '" + pwd + "'";        Statement stmt = connection.createStatement();        ResultSet rs = stmt.executeQuery(sqlString);    **if** (!rs.next()) {  **throw** **new** SecurityException(            "User name or password incorrect"          );        }          // Authenticated; proceed      } **finally** {  **try** {          connection.close();        } **catch** (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses a parametric query with a ? character as a placeholder for the argument. This code also validates the length of the username argument, preventing an attacker from submitting an arbitrarily long user name. |
| **public** **void** doPrivilegedAction(    String username, **char**[] password  ) **throws** SQLException {    Connection connection = getConnection();  **if** (connection == **null**) {      // Handle error    }  **try** {      String pwd = hashPassword(password);        // Validate username length  **if** (username.length() > 8) {        // Handle error      }        String sqlString =        "select \* from db\_user where username=? and password=?";      PreparedStatement stmt = connection.prepareStatement(sqlString);      stmt.setString(1, username);      stmt.setString(2, pwd);      ResultSet rs = stmt.executeQuery();  **if** (!rs.next()) {  **throw** **new** SecurityException("User name or password incorrect");      }        // Authenticated; proceed    } **finally** {  **try** {        connection.close();      } **catch** (SQLException x) {        // Forward to handler      }    }  } |

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

| **Principles(s):** Use the set\*() methods of the PreparedStatement class to enforce strong type checking. This technique mitigates the SQL injection vulnerability because the input is properly escaped by automatic entrapment within double quotes. Note that prepared statements must be used even with queries that insert data into the database. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| | **Tool** | **Version** | **Checker** | **Description** | | --- | --- | --- | --- | | [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) |  |  |  | | 2.1.3 | **Tainting Checker** | Trust and security errors |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **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-CON32-C] | Prevent data races when accessing bit-fields from multiple threads |

| **Noncompliant Code** |
| --- |
| Adjacent bit-fields may be stored in a single memory location. Consequently, modifying adjacent bit-fields in different threads is undefined behavior, as shown in this noncompliant code example: |
| **struct** multi\_threaded\_flags {    unsigned **int** flag1 : 2;    unsigned **int** flag2 : 2;  };    **struct** multi\_threaded\_flags flags;    **int** thread1(**void** \*arg) {    flags.flag1 = 1;  **return** 0;  }    **int** thread2(**void** \*arg) {    flags.flag2 = 2;  **return** 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution protects all accesses of the flags with a mutex, thereby preventing any data races: |
| #include <threads.h>    **struct** multi\_threaded\_flags {    unsigned **int** flag1 : 2;    unsigned **int** flag2 : 2;  };    **struct** mtf\_mutex {  **struct** multi\_threaded\_flags s;    mtx\_t mutex;  };    **struct** mtf\_mutex flags;    **int** thread1(**void** \*arg) {  **if** (thrd\_success != mtx\_lock(&flags.mutex)) {      /\* Handle error \*/    }    flags.s.flag1 = 1;  **if** (thrd\_success != mtx\_unlock(&flags.mutex)) {      /\* Handle error \*/    }  **return** 0;  }    **int** thread2(**void** \*arg) {  **if** (thrd\_success != mtx\_lock(&flags.mutex)) {      /\* Handle error \*/    }    flags.s.flag2 = 2;  **if** (thrd\_success != mtx\_unlock(&flags.mutex)) {      /\* Handle error \*/    }  **return** 0;  } |

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

| **Principles(s):** It is almost certain that flag1 and flag2 are stored in the same word. Using a compiler that conforms to C99 or earlier, if both assignments occur on a thread-scheduling interleaving that ends with both stores occurring after one another, it is possible that only one of the flags will be set as intended. The other flag will contain its previous value because both members are represented by the same word, which is the smallest unit the processor can work on. Before the changes were made to the C Standard for C11, there were no guarantees that these flags could be modified concurrently. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | **read\_data\_race**  **write\_data\_race** | Supported by sound analysis (data race alarm) |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-CON32** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **CONCURRENCY.DATARACE CONCURRENCY.MAA** | Data race Multiple Accesses of Atomic |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | |  |  |  | | --- | --- | --- | |  | **C1774, C1775** |  | |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-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: |
| #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: |
| **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):** Static assertions allow incorrect assumptions to be diagnosed at compile time instead of resulting in a silent malfunction or runtime error. Because the assertion is performed at compile time, no runtime cost in space or time is incurred. An assertion can be used at file or block scope, and failure results in a meaningful and informative diagnostic error message. |
| --- |

**Threat Level**

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

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-ERR55-CPP] | Honor exception specifications |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a function is declared as nonthrowing, but it is possible for std::vector::resize() to throw an exception when the requested memory cannot be allocated. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {    v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function's noexcept-specification is removed, signifying that the function allows all exceptions. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) {    v.resize(s); // May throw, but that is okay  } |

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

| **Principles(s):** Some vendors provide language extensions for specifying whether a function throws. For instance, Microsoft Visual Studio provides \_\_declspec(nothrow)), and Clang supports \_\_attribute\_\_((nothrow)). Currently, the vendors do not document the behavior of specifying a nonthrowing function using these extensions. Throwing from a function declared with one of these language extensions is presumed to be undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **unhandled-throw-noexcept** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-ERR55** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **LANG.STRUCT.EXCP.THROW** | Use of throw |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | |  |  |  | | --- | --- | --- | |  | **C++4035, C++4036, C++4632** |  | |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input output | [STD-008-CPP] | Do not alternately input and output from a file stream without an intervening positioning call. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example appends data to the end of a file and then reads from the same file. However, because there is no intervening positioning call between the formatted output and input calls, the behavior is undefined. |
|  |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::basic\_istream::seekg() function is called between the output and input, eliminating the undefined behavior. |
|  |

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

| **Principles(s):** : Alternately inputting and outputting from a stream without an intervening flush or positioning call is undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| Tool | Version | Checker | Description Tool |
| --- | --- | --- | --- |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP\_FI)50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
|  |  |  |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programing | [STD-009-CPP] | Do not invoke virtual functions from constructors or destructors |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the base class attempts to seize and release an object's resources through calls to virtual functions from the constructor and destructor. However, the B::B() constructor calls B::seize() rather than D::seize(). Likewise, the B::~B() destructor calls B::release() rather than D::release(). |
|  |

| **Compliant Code** |
| --- |
| In this compliant solution, the constructors and destructors call a nonvirtual, private member function (suffixed with mine) instead of calling a virtual function. The result is that each class is responsible for seizing and releasing its own resources. |
|  |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC++OOP50** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.1 | |  |  |  | | --- | --- | --- | |  | **C++4035, C++4036, C++4632** |  | |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | [STD-010-CCP] | Value returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the programmer forgot to return the input value for positive input, so not all code paths return a value. |
|  |

| **Compliant Code** |
| --- |
| In this compliant solution, all code paths now return a value. |
|  |

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

| **Principles(s):** Failing to return a value from a code path in a value-returning function results in undefined behavior that might be exploited to cause data integrity violations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | Return-implicit | Fully Checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC++MSC52** | Implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | LANG.STRUCT.MRS | Missing return statement |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2021.1 |  |  |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

[Insert your written explanations here.]

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-PRE30-C | LOW | Unlikely | Medium | P2 | L3 |
| STD-INT30-C | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-IDS00-J | High | Probable | Medium | P12 | L1 |
| STD-CON32-C | Medium | Probable | Medium | P8 | L2 |
| STD-DCL03-C | Low | Unlikely | High | P1 | L3 |
| STD-ERR55-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CCP | Low | Likely | Medium | P6 | L2 |
| STD-009-CCP | Low | Likely | Medium | P2 | L3 |
| STD-010-CCP | Medium | Probable | Medium | P8 | L2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption in rest refers to encrypted data stored that is house physically in a computer. |
| Encryption at flight | Encryption at flight refers to data in transit, typically in a computer network. |
| Encryption in use | Encryption in use refers to data story in memory, CPU and CPU registry. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This policy is important in order to determine who has access to data. This ensure the correct user is accessing data. |
| Authorization | This policy is important in order allow access to data with certain privileges and permissions. |
| Accounting | This policy is used to record access to data. |

**\***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 | 10/21/2022 | Policy Update | George Harrison |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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