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



Green Pace Secure Development Policy

Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

# Overview

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

# Purpose

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

# Scope

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

# Module Three Milestone

## Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Always validate input from untrusted sources. If you don’t validate input, there is potential for a vulnerability, such as an SQL Injection. |
| 1. Heed Compiler Warnings | Pay attention to compiler warnings. You should use the highest warning level for your compiler to compile your code. |
| 1. Architect and Design for Security Policies | Design software to implement and enforce security policies. |
| 1. Keep It Simple | Keep the design as simple as possible. More complicated systems will increase the likelihood that errors will be made as well as increase the time and money needed to better secure that system. |
| 1. Default Deny | Base access on permissions and design a scheme for which conditions access is granted. |
| 1. Adhere to the Principle of Least Privilege | Each process should execute with the least amount of privilege necessary. If a certain program only needs access to non-sensitive data, don’t grant access to sensitive data. |
| 1. Sanitize Data Sent to Other Systems | Using off the shelf components can bring in a vulnerability by allowing an attacker to utilize unused functionality. |
| 1. Practice Defense in Depth | Never rely on a single defense strategy. If an attacker manages to break through one layer, there are additional layers that can stop them. |
| 1. Use Effective Quality Assurance Techniques | Thoroughly testing your program and even bringing in an outside source to test the program can help bring up any vulnerabilities before the program reaches the public. The more people stress testing the system and reviewing code, the better. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for any program you work on. |

## 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-DAT-001 | Guarantee that library functions do not overflow |

| **Noncompliant Code** |
| --- |
| Std::copy() provides no inherent bounds checking and can lead to a buffer overflow. The below code example copies a vector of integers from src to dest. Because std::copy() does nothing to expand dest, the program will overflow. |
| #include <algorithm>  #include <vector>    void f(const std::vector<**int**> &src) {    std::vector<**int**> dest;    std::copy(src.begin(), src.end(), dest.begin());    // ...  } |

| **Compliant Code** |
| --- |
| The below code ensures the container being copied to can hold all the elements being copied. The solution enlarges the capacity of the vector prior to the copy operation. |
| #include <algorithm>  #include <vector>  void f(const std::vector<**int**> &src) {    // Initialize dest with src.size() default-inserted elements    std::vector<**int**> dest(src.size());    std::copy(src.begin(), src.end(), dest.begin());    // ...  } |

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

|  |
| --- |
| Validate Input Data – If you assume data being passes is always going to be within the range you expect, you open yourself up for an overflow situation. In the above example, you anticipate a situation in which the data is greater than what you’re needing so you check the size first in order to resize if needed. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | Invalid\_pointer\_dereference |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.1p0 | BADFUNC.BO.\*  LANG.MEM.BO | A collection of warning classes that report uses of library functions prone to internal buffer overflows.  Buffer Overrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++3526, C++3527, C++3528, C++3529, C++3531, C++3532, C++3533, C++3534 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-CTR52-a | Do not pass empty container iterators to std algorithms as destinations |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-DAT-002 | Value-returning functions must return a value from all exit paths. |

| **Noncompliant Code** |
| --- |
| In the below example, the return statement doesn’t handle positive input so not all code paths return a value. |
| **int** absolute\_value(**int** a) {    if (a < 0) {      return -a;    }  } |

| **Compliant Code** |
| --- |
| The below example returns all code paths. |
| **int** absolute\_value(**int** a) {    if (a < 0) {      return -a;    }    return a;  } |

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

|  |
| --- |
| Validate input/ use effective quality assurance techniques – Testing your code thoroughly can prevent mistakes like the one shown above. Making sure all forms of input can produce a return value. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | Return-implicit | Fully Checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.1p0 | LANG.STRUCT.MRS | Missing return statement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-MSC52-a | All exit paths from a function with non-void return type shall have an explicit return statement with an expression |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021a | CERT C++:MSC52-CPP | Checks for missing return statements (rule partially covered) |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-STR-001 | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| The below example has unbounded input which could lead to a buffer overflow |
| #include <iostream>    void f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| To prevent a buffer overflow, use std::string instead of a bounded array. |
| #include <iostream>  #include <string>    void f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

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

|  |
| --- |
| Validate input / heed compiler warnings – The above example has unbounded input which could result in a buffer overflow. Making sure the input is only the size desired can prevent this. Also, paying attention to compiler warnings can help you catch something you missed. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.1p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the ‘char’ buffer to store input from ‘std::cin’ |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021a | CERT C++:STR50-CPP | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation   Rule partially covered |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++2835, C++2836, C++2839, C++5216 |  |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-SQL-001 | Prevent SQL Injection |

| **Noncompliant Code** |
| --- |
| The below example permits a SQL injection attack by incorporating the unsanitized input argument username into the SQL command which would allow an attacker ‘1’ = ‘1’ which would always be true. |
| 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 below example uses a parametric query with a ? character as a placeholder for the argument. The length of username is also validated, preventing an attacker from submitting an arbitrarily long username. |
| 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.**

|  |
| --- |
| **Validate input / sanitize data – Trusting the user to input only what you expect can lead to an SQL injection. Checking the input before passing the SQL command can prevent an SQL injection.** |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | Tainting Checker | Trust and security errors (see chapter 8) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2021.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 6.7 | S2077  S3649 | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks. |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-MEM-001 | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| In the below example, an array of int is created using ::operator new[] (std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because :operator new[] (std::size\_t) can throw an exception if the allocation fails, it could lead to abnormal termination of the program |
| #include <cstring>    void f(const **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = new **int**[size];    std::**memcpy**(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    void f(const **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = new (std::nothrow) **int**[size];    if (!copy) {      // Handle error      return;    }    std::**memcpy**(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

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

|  |
| --- |
| Heed compiler warnings – Pay attention to any warnings from the compiler such as a function that is marked noexcept that throws an exception. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Coverity) | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function’s parameters is undefined |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++3225, C++3226, C++3227, C++3228, C++3229, C++4632 |  |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021a | CERT C++:MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-AST-001 | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| In the below example, the call to f (), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func () may throw an exception. |
| #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** |
| --- |
| F() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #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.**

|  |
| --- |
| **Use effective quality assurance techniques – testing and running your code can see if the program terminates when it’s not supposed to.** |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 122 S | Enhanced Enforcement |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2021.1 | MISRA.CATCH.ALL  CERT.ERR.ABRUPT\_TERM |  |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021a | CERT C++:ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered) |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-EXC-001 | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In the below example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| The main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    **int** main() {    try {      f();    } catch (...) {      // Handle error    }  } |

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

|  |
| --- |
| Use effective quality assurance techniques – Make sure your code handles all exceptions. |

**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) | 20.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | **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 |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021a | CERT C++:ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | 4035, 4036, 4037 |  |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | STD-INT-001 | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| In the below example, the program attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. On a two’s complement system, the valid range of values that can be represented by EnumType are [0..3] so if a value outside of that range were passed to f(), the cast to EnumType would result in a unspecified value, and use that value within the if statement results in unspecified behavior. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does this by restring the converted value to one for which there is a specific enumerator value. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

|  |
| --- |
| Validate input – Check your data before trying to use it. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-INT50 |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | C++3013 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | 3013 |  |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declarations | STD-DCL-001 | Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| A common practice is to use a macro in a preprocessor conditional that guards against multiple inclusions of a header file. While this is a recommended practice, many programs use reserved names as the header guards. Such a name may clash with reserved names defined by the implementation of the C++ standard template library in its headers or with reserved names implicitly predefined by the compiler even when no C++ standard library header is included. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| This solution avoids using leading or trailing underscores in the name of the header guard. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

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

|  |
| --- |
| **Heed compiler warnings / keep it simple – Pay attention to the naming of headers as they may be already reserved.** |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreserved-id-macro  -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Wevertying. This flag does not catch all instances of this rule, such as redefining reserved names. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.1p0 | LANG.ID.NU.MK  LANG.STRUCT.DECL.RESERVED | Macro named is C keyword  Declaration of reserved name |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-DCL51-a  CERT\_CPP-DCL51-b  CERT\_CPP-DCL51-c  CERT\_CPP-DCL51-d  CERT\_CPP-DCL51-e  CERT\_CPP-DCL51-f | Do not #define or #undef identifiers with names which start with underscore  Do not redefine reserved words  Do not #define nor #undef identifier ‘defined’  The names of standard library macros, objections and functions shall not be reused  The names of standard library macros, objects and functions shall not be reused (C90)  The names of standard library macros, objections and functions shall not be reused (C99) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021a | CERT C++:DCL51-CPP | Checks for redefinitions of reserved identifiers (rule partially covered) |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | STD-CTR-001 | Guarantee that container indices and iterators are within the valid range. |

| **Noncompliant Code** |
| --- |
| The below example shows a function, insert\_in\_table(), that has two int paramters, pos and value, both of which can be influenced by data originating from untrusted sources. The function performs a range check to ensure that pos does not exceed the upper bound of the array, specified by tableSize, but fails to check the lower bound. Because pos is declared as a signed int, this parameter can assume a negative value, resulting in a write outside the bounds of the memory referenced by table. |
| #include <cstddef>    void insert\_in\_table(**int** \*table, std::**size\_t** tableSize, **int** pos, **int** value) {    if (pos >= tableSize) {      // Handle error      return;    }    table[pos] = value;  } |

| **Compliant Code** |
| --- |
| In this solution, the parameter pos is declared as size\_t, which prevents the passing of negative arguments. |
| #include <cstddef>    void insert\_in\_table(**int** \*table, std::**size\_t** tableSize, std::**size\_t** pos, **int** value) {    if (pos >= tableSize) {      // Handle error      return;    }    table[pos] = value;  } |

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

|  |
| --- |
| **Use effective quality assurance techniques – Testing code and having code reviews can detect and prevent errors such as the one in the noncompliant code example.** |

**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=222953724) | 20.10 | Overflow\_upon\_dereference |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TO  LANG.MEM.TU  LANG.MEM.TBA  LANG.STRUCT.PBB  LANG.STRUCT.PPE | Buffer overrun  Buffer underrun  Type overrun  Type underrun  Tainted buffer access  Pointer before beginning of object  Pointer past end of object |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.1 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021a | CERT C++:CTR50-CPP | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

## 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.

Automated testing needs to be implemented at the very beginning and run daily, especially on newly committed code. You do not want to wait until right before production to test your code for vulnerabilities. Also, developers should be testing their code as they write it, scanning for vulnerabilities before ever committing. It might even be beneficial to do penetration testing before the code is released, as well as after it is released by in house pen testers as well as outside pen testers.

## 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-AST-001 | Low | Probable | Medium | P4 | L3 |
| STD-CTR-001 | High | Likely | High | P9 | L2 |
| STD-DAT-001 | High | Likely | Medium | P18 | L1 |
| STD-DAT-002 | Medium | Probable | Medium | P8 | L2 |
| STD-DCL-001 | Low | Unlikely | Low | P3 | L3 |
| STD-EXC-001 | Low | Probable | Medium | P4 | L3 |
| STD-INT-001 | Medium | Unlikely | Medium | P4 | L3 |
| STD-MEM-001 | High | Likely | Medium | P18 | L1 |
| STD-SQL-001 | High | Probable | Medium | P12 | L1 |
| STD-STR-001 | High | Likely | Medium | P18 | L1 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

## 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 | Encrypting the data that is stored on disc. This ensures that if someone breaks in, they don’t can’t use the data as it’s unreadable. They will need to correct key to utilize the data they stole. |
| Encryption at flight | Encrypting data as it moves across the network. As data passes across the network, it is vulnerable to a man in the middle attack where a hacker can grab that data so you need to make sure it’s protected in case they do get it. |
| Encryption in use | Encrypting the data in memory. There are methods of dumping data that is in use and if they’re un-encrypted, that sensitive data will be accessed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Identifying a user and ensuring they are who they say they are. This can be done in several different ways but the most common is through unique username/password combinations. Once the user has been authenticated, only then will they be granted access into the system. |
| Authorization | Each user should be assigned an authorization level, giving them the bare minimum that is required for that role. A user shouldn’t have the same level of authorization in the system as an administrator. By doing this, if a user’s credentials are compromised, an intruder doesn’t get free reign into the system. |
| Accounting | Keeping a log of activity by the user. What did they access? When did they access it? |

**\***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 |  |
| 2.0 | 08/14/2021 | Final Design | Alan Graves |  |
|  |  |  |  |  |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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