**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 | Input data should always be validated. Input validation can eliminate many software vulnerabilities. Without input validation malicious users can take advantage of this by command line arguments and external data sources. |
| 1. Heed Compiler Warnings | Compiler warnings should not be ignored as they can lead to possible vulnerabilities or cascading bugs in the future that can be exploited. Use coding best practices to modify the code to eliminate compiler warnings, also use static and dynamic analysis/testing tools to detect and eliminate additional security vulnerabilities. |
| 1. Architect and Design for Security Policies | Software architecture should be designed around enforcing security policies. If the system being created requires different privileges at different times, the system should be divided into distinct subsystems, each with their appropriate privilege set. |
| 1. Keep It Simple | The design should be kept as simple as possible, as complex designs increase the risk of errors in implementation and configuration. Complex systems also require more security mechanisms to achieve the same security of a simple system. |
| 1. Default Deny | By default, access should be denied, and the security mechanisms should identify conditions for permission rather than exclusion. |
| 1. Adhere to the Principle of Least Privilege | The principle of lease privilege recommends only allowing enough access to a user to perform the required job. This reduces the risk of attackers gaining access to important systems or sensitive data. |
| 1. Sanitize Data Sent to Other Systems | All data passed to complex subsystems such as command shells and relational databases should be sanitized or erased. This is because attackers may be able to invoke unused functionality in these components by using SQL Injection attacks or other malicious techniques. |
| 1. Practice Defense in Depth | Multiple defense strategies should be put in place so that if one layer of defense fails, another layer can cover the previous layer’s security flaws, preventing them from being exploitable. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance techniques are essential for identifying and eliminating security vulnerabilities. Effective quality assurance would include techniques such as Fuzz testing, penetration testing, and source code audits. |
| 1. Adopt a Secure Coding Standard | Depending on the development language and platform being utilized, secure coding standards and best practices should be applied for each situation. |

### 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** | **Do Not Define a C-style variadic function** |
| --- | --- | --- |
| **Data Type** | [STD-DCL50-CPP] | Functions can be defined to accept more formal arguments at the call site than are specified by the parameter declaration clause. Such functions are called variadic functions because they can accept a variable number of arguments from a caller. C++ provides two mechanisms by which a variadic function can be defined: function parameter packs and use of a C-style ellipsis as the final parameter declaration. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses a C-style variadic function to add a series of integers together. The function reads arguments until the value 0 is found. Calling this function without passing the value 0 as an argument (after the first two arguments) results in undefined behavior. Furthermore, passing any type other than an int also results in undefined behavior. |
| #include <cstdarg>    int add(int first, int second, ...) {  int r = first + second;  va\_list va;  va\_start(va, second);  while (int v = va\_arg(va, int)) {  r += v;  }  va\_end(va);  return r;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, a variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. |
| #include <type\_traits>    template <typename Arg, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Arg s) { return f + s; }    template <typename Arg, typename... Ts, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg f, Ts... rest) {  return f + add(rest...);  } |

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

| **Principles(s):**  **Heed Compiler Warnings:** This coding standard is relevant because it emphasizes the importance of paying attention to compiler warnings, as doing so could lead to issues or vulnerabilities related to improper use of variadic functions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Function-ellipsis | Fully Checked |
| LDRA tool suite | 9.7.1 | 41 S | Fully Implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-DCL50-a | Functions shall not be defined with a variable number of arguments |
| RuleChecker | 22.10 | Function-ellipsis | Fully Checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not rely on the value of a moved-from object** |
| --- | --- | --- |
| **Data Value** | [STD-EXP63-CPP] | Many types, including user-defined types and types provided by the Standard Template Library, support move semantics. Except in rare circumstances, an object of a type that supports move operations will be left in a valid, but unspecified state after the object's value has been moved. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the integer values 0 through 9 are expected to be printed to the standard output stream from a std::string rvalue reference. However, because the object is moved and then reused under the assumption its internal state has been cleared, unexpected output may occur despite not triggering undefined behavior. |
| #include <iostream>  #include <string>    void g(std::string v) {  std::cout << v << std::endl;  }    void f() {  std::string s;  for (unsigned i = 0; i < 10; ++i) {  s.append(1, static\_cast<char>('0' + i));  g(std::move(s));  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::string object is initialized to the expected value on each iteration of the loop. This practice ensures that the object is in a valid, specified state prior to attempting to access it in g(), resulting in the expected output. |
| #include <iostream>  #include <string>    void g(std::string v) {  std::cout << v << std::endl;  }    void f() {  for (unsigned i = 0; i < 10; ++i) {  std::string s(1, static\_cast<char>('0' + i));  g(std::move(s));  }  } |

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

| **Principles(s):**  **Heed Compiler Warnings:** Ignoring compiler warnings could lead to issues or vulnerabilities related to incorrect usage of moved objects.  **Keep It Simple:** Complex designs can increase the likelihood of relying on moved-from objects. Simplicity in design can help to avoid unintentional dependencies on the state of moved objects. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | LANG.MEM.NPD  LANG.MEM.UVAR | Null Pointer Dereference  Uninitialized Variable |
| Helix QAC | 2023.3 | DF4701, DF4702, DF4703 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-EXP63-a | Do not rely on the value of a moved-from object |
| Polyspace Bug Finder | 7.26 | V1030 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Detect errors when converting a string to a number** |
| --- | --- | --- |
| **String Correctness** | [STD-ERR62-CPP] | The process of parsing an integer or floating-point number from a string can produce many errors. These error conditions must be detected and addressed when a string-to-number conversion is performed using a formatted input stream. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, multiple numeric values are converted from the standard input stream. However, if the text received from the standard input stream cannot be converted into a numeric value that can be represented by an int, the resulting value stored into the variables i and j may be unexpected. |
| #include <iostream>    void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, exceptions are enabled so that any conversion failure results in an exception being thrown. However, this approach cannot distinguish between which values are valid and which values are invalid and must assume that all values are invalid. |
| #include <iostream>    void f() {  int i, j;    std::cin.exceptions(std::istream::failbit | std::istream::badbit);  try {  std::cin >> i >> j;  // ...  } catch (std::istream::failure &E) {  // Handle error  }  } |

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

| **Principles(s):**  **Heed Compiler Warnings:** When converting a string to a number, ignoring compiler warnings related to data type mismatches can lead to errors and vulnerabilities.  **Use Effective Quality Assurance Techniques:** Code reviews can help identify issues related to error handling during conversion. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| CodeSonar | 7.4p0 | **BADFUNC.ATOF** **BADFUNC.ATOI** **BADFUNC.ATOL** **BADFUNC.ATOLL** | Use of atof Use of atoi Use of atol Use of atoll |
| Helix QAC | 2023.3 | C++ 3161 |  |
| Parasoft C/C++ test | 2023.1 | **CERT\_CPP-ERR62-a** | The library functions atof, atoi and atol from library stdlib.h shall not be used. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-IDS00-J] | SQL injection vulnerabilities arise in applications where elements of a SQL query originate from an untrusted source. Without precautions, the untrusted data may maliciously alter the query, resulting in information leaks or data modification. The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as parameterized queries and stored procedures. |

| **Noncompliant Code** |
| --- |
| The prepared statement still permits a SQL injection attack by incorporating the unsanitized input argument username into the prepared statement. |
| class Login {  public Connection getConnection() throws SQLException {  DriverManager.registerDriver(new  com.microsoft.sqlserver.jdbc.SQLServerDriver());  String dbConnection =  PropertyManager.getProperty("db.connection");  // Can hold some value like  // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  return DriverManager.getConnection(dbConnection);  }    String hashPassword(char[] password) {  // Create hash of password  }    public void doPrivilegedAction(  String username, char[] password  ) throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);  String sqlString = "select \* from db\_user where username=" +  username + " and password =" + pwd;  PreparedStatement stmt = connection.prepareStatement(sqlString);    ResultSet rs = stmt.executeQuery();  if (!rs.next()) {  throw new SecurityException("User name or password incorrect");  }  // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  } |

| **Compliant Code** |
| --- |
| This 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. |
| class Login {  public Connection getConnection() throws SQLException {  DriverManager.registerDriver(new  com.microsoft.sqlserver.jdbc.SQLServerDriver());  String dbConnection =  PropertyManager.getProperty("db.connection");  // Can hold some value like  // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  return DriverManager.getConnection(dbConnection);  }    String hashPassword(char[] password) {  // Create hash of password  }    public void doPrivilegedAction(  String username, char[] password  ) throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);  String sqlString = "select \* from db\_user where username=" +  username + " and password =" + pwd;  PreparedStatement stmt = connection.prepareStatement(sqlString);    ResultSet rs = stmt.executeQuery();  if (!rs.next()) {  throw new SecurityException("User name or password incorrect");  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  } |

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

| **Principles(s):**  **Validate Input Data:** SQL injection often occurs due to improper handling of user input, and input validation is key in preventing these vulnerabilities.  **Sanitize Data Sent to Other Systems:** Sanitizing data and ensuring that user input does not directly impact SQL queries is a fundamental step in preventing SQL injection. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| CodeSonar | 7.4p0 | JAVA.IO.INJ.SQL | SQL Injection |
| Coverity | 7.5 | **SQLI** **FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| FindBugs | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Prevent data races when accessing bit-fields from multiple threads** |
| --- | --- | --- |
| **Memory Protection** | [STD-CON52-CPP] | When accessing a bit-field, a thread may inadvertently access a separate bit-field in adjacent memory. |

| **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 MultiThreadedFlags {  unsigned int flag1 : 2;  unsigned int flag2 : 2;  };    MultiThreadedFlags flags;    void thread1() {  flags.flag1 = 1;  }    void thread2() {  flags.flag2 = 2;  } |

| **Compliant Code** |
| --- |
| This compliant solution protects all accesses of the flags with a mutex, thereby preventing any data races. |
| #include <mutex>    struct MultiThreadedFlags {  unsigned int flag1 : 2;  unsigned int flag2 : 2;  };    struct MtfMutex {  MultiThreadedFlags s;  std::mutex mutex;  };    MtfMutex flags;    void thread1() {  std::lock\_guard<std::mutex> lk(flags.mutex);  flags.s.flag1 = 1;  }    void thread2() {  std::lock\_guard<std::mutex> lk(flags.mutex);  flags.s.flag2 = 2;  } |

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

| **Principles(s):**  **Heed Compiler Warnings:** Compiler warnings related to thread safety and data races can help identify potential issues when accessing bit-fields from multiple threads.  **Use Effective Quality Assurance Techniques:** Quality assurance techniques such as code reviews can help identify concurrency-related issues.  **Adopt a secure coding standard:** Adhering to secure coding standards and best practices can help ensure that thread safety issues related to bit-field access are addressed as part of the coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.1 | **read\_write\_data\_race** **write\_write\_data\_race** | Supported |
| CodeSonar | 7.4p0 | |  |  | | --- | --- | | CONCURRENCY.DATARACE |  | | Data Race |
| Coverity | 6.5 | RACE\_CONDITION | Fully Implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-CON52-A | Use locks to prevent race conditions when modifying bit fields. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | [STD-DCL03-C] | Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities. |

| **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):**  **Heed Compiler Warnings:** Paying attention to compiler warnings is essential to ensure that static assertions and constant expressions are used correctly.  **Use Effective Quality Assuraance Techniques:** Effective quality assurance techniques such as code reviews and testing, can help ensure that static assertions are properly implemented and that constant expressions are validated. |
| --- |

**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 |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Guarantee exception safety** |
| --- | --- | --- |
| **Exceptions** | [STD-ERR56-CPP] | Proper handling of errors and exceptional situations is essential for the continued correct operation of software. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a flawed copy assignment operator. The implicit invariants of the class are that the array member is a valid (possibly null) pointer and that the nElems member stores the number of elements in the array pointed to by array. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }      IntArray(const IntArray& that); // nontrivial copy constructor  IntArray& operator=(const IntArray &rhs) {  if (this != &rhs) {  delete[] array;  array = nullptr;  nElems = rhs.nElems;  if (nElems) {  array = new int[nElems];  std::memcpy(array, rhs.array, nElems \* sizeof(\*array));  }  }  return \*this;  }    // ...  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the copy assignment operator provides the strong exception safety guarantee. The function allocates new storage for the copy before changing the state of the object. Only after the allocation succeeds does the function proceed to change the state of the object. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }    IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {  int \*tmp = nullptr;  if (rhs.nElems) {  tmp = new int[rhs.nElems];  std::memcpy(tmp, rhs.array, rhs.nElems \* sizeof(\*array));  }  delete[] array;  array = tmp;  nElems = rhs.nElems;  return \*this;  }    // ...  }; |

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

| **Principles(s):**  **Keep It Simple:** Keeping error handling code simple and straightforward can make it easier to ensure exception safety.  **Use Effective Quality Assurance Techniques:** Testing and code reviews are vital for ensuring that exception handling and error handling mechanisms are implemented correctly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | **ALLOC.LEAK** | Leak |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S, 56 D, 71 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR56-a** **CERT\_CPP-ERR56-b** | Always catch exceptions Do not leave 'catch' blocks empty |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR56-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr56cpp.html) | Checks for exceptions violating class invariant (rule fully covered). |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Limit the lifetime of sensitive data** |
| --- | --- | --- |
| Data | [STD-MSC59-J] | Sensitive data in memory can be vulnerable to compromise. An adversary who can execute code on the same system as an application may be able to access such data. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses a BufferedReader to wrap an InputStreamReader object so that sensitive data can be read from a file. |
| void readData() throws IOException {  BufferedReader br = new BufferedReader(new InputStreamReader(  new FileInputStream("file")));  // Read from the file  String data = br.readLine();  } |

| **Compliant Code** |
| --- |
| This compliant solution uses a directly allocated NIO (new I/O) buffer to read sensitive data from the file. The data can be cleared immediately after use and is not cached or buffered at multiple locations. It exists only in the system memory. |
| void readData() {  int bufferSize = 16 \* 1024;  byte zeroes = new byte[bufferSize];  ByteBuffer buffer = ByteBuffer.allocateDirect(bufferSize);  try (FileChannel rdr = (new FileInputStream("file")).getChannel()) {  while (rdr.read(buffer) > 0) {    // Do something with the buffer    buffer.clear();  buffer.put(zeroes); // overwrite buffer with zeroes  buffer.clear();  }  } catch (Throwable e) {  // Handle error  }  } |

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

| **Principles(s):**  **Sanitized Data Sent to Other Systems:** This principle is relevant as it includes ensuring that sensitive data is properly handled and protected.  **Use Effective Quality Assurance Techniques:** Effective quality assurance techniques such as code reviews and testing, are essential in ensuring that sensitive data is handled securely. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Hight | Probable | Moderate | High | Critical |

**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) | **7.4p0** | ALLOC.UAF | Implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MEM50-a** | implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| Memory | [STD-MEM50-CPP] | Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable vulnerabilities. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):**  **Heed Compiler Warnings:** Compiler warnings can often assist in identifying potential issues related to accessing freed memory.  **Keep It Simple:** Keeping the code as simple as possible can help reduce the likelihood of complex memory management erros, including accessing greed memory. |
| --- |

**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) | **7.4p0** | ALLOC.UAF | **Use after free** |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |

#### Coding Standard 10

| **Coding Standard** | **Label** | **When Data must be accessed by multiple threads, provide a mutex and guarantee no adjacent data is also accessed.** |
| --- | --- | --- |
| Data | [STD-POS49-C] | When multiple threads must access or make modifications to a common variable, they may also inadvertently access other variables adjacent in memory. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, two executing threads simultaneously access two separate members of a global struct. |
| struct multi\_threaded\_flags {  unsigned int flag1 : 2;  unsigned int flag2 : 2;  };    struct multi\_threaded\_flags flags;    void thread1(void) {  flags.flag1 = 1;  }    void thread2(void) {  flags.flag2 = 2;  } |

| **Compliant Code** |
| --- |
| This compliant solution protects all accesses of the flags with a mutex, thereby preventing any thread-scheduling interleaving from occurring. |
| struct multi\_threaded\_flags {  volatile unsigned int flag1 : 2;  volatile unsigned int flag2 : 2;  };    union mtf\_protect {  struct multi\_threaded\_flags s;  long padding;  };    static\_assert(sizeof(long) >= sizeof(struct multi\_threaded\_flags));    struct mtf\_mutex {  union mtf\_protect u;  pthread\_mutex\_t mutex;  };    struct mtf\_mutex flags;    void thread1(void) {  int result;  if ((result = pthread\_mutex\_lock(&flags.mutex)) != 0) {  /\* Handle error \*/  }  flags.u.s.flag1 = 1;  if ((result = pthread\_mutex\_unlock(&flags.mutex)) != 0) {  /\* Handle error \*/  }  }    void thread2(void) {  int result;  if ((result = pthread\_mutex\_lock(&flags.mutex)) != 0) {  /\* Handle error \*/  }  flags.u.s.flag2 = 2;  if ((result = pthread\_mutex\_unlock(&flags.mutex)) != 0) {  /\* Handle error \*/  }  } |

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

| **Principles(s):**  **Heed Compiler Warnings:** Compiler warnings can hel0p identify potential issues related to concurrent access to data.  **Keep It Simple:** Simplicity can make it easier to guarantee that adjacent data is not inadvertently accessed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **CONCURRENCY.DATARACE** **CONCURRENCY.MAA** | Data race Multiple Accesses of Atomic |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-POS49-a** | Use locks to prevent race conditions when modifying bit fields |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **457** | Partially supported: access is detected at the object level (not at the field level) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule POS49-C](https://www.mathworks.com/help/bugfinder/ref/certcrulepos49c.html) | Checks for data race (rule fully 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 with 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.

Automation should be utilized throughout the DevOps process to increase overall effectiveness and efficiency. During code development coding standards should be enforced using automated code analysis tools to ensure best practices.

During the testing phase as well as throughout production, automated test suites for unit tests, integration tests, and regression tests should be used to assure quality and secure code.

During the monitoring phase, automated monitoring solutions should be implemented that trigger alerts based on predefined conditions, as well as automated log management with the use of automated log analysis tools to detect and diagnose issues detected by the automated real-time monitoring.

Automated alerting and incident triage should be used to enhance the response aspect of the DevSecOps. With automated alerting and incident triage, automated alerts for incidents can be set up and can be used to alert the appropriate teams for efficient responses.

### 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-IDS00-J | High | Likely | Medium | P18 | L1 |
| STD-DCL03-C | Low | Unlikely | High | P1 | L3 |
| STD-POS49-C | Medium | Probable | Medium | P8 | L2 |
| STD-DCL50-CPP | High | Probable | Medium | P12 | L1 |
| STD-MEM50-CPP | High | Likely | Medium | P18 | L1 |
| STD-CON52-CPP | Medium | Probably | Medium | P8 | L2 |
| STD-ERR56-CPP | High | Likely | High | P9 | L2 |
| STD-MSC59-J | Hight | Probable | Moderate | High | Critical |
| STD-ERR62-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-EXP63-CPP | 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 at rest refers to the encryption of data when it is stored in a database, file, system, or storage device. The policy mandates that all sensitive data stored in databases or storage must be encrypted using strong encryption algorithms. This policy applies to all data that is stored in any digital format, to protect data from unauthorized access in case of data breaches. |
| Encryption at flight | Encryption in flight is also known as data in transit encryption. It involves securing data as it travels between systems, over networks, or during communication. The policy mandates the use of encryption protocols for all data transmitted over public or untrusted networks. It ensures data confidentiality during transmission and guards against eavesdropping and interception. |
| Encryption in use | Encryption in use focuses on protecting data while it is actively being processed or used by applications or users. The policy requires the use of encryption techniques such as secure memory handling, application-level encryption, and hardware security modules for sensitive data during processing. This policy applies to applications that handle sensitive data, it ensures data confidentiality even when data is being actively processed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to the process of verifying the identity of users, systems, or entities attempting to access resources or systems. The policy mandates the use of strong authentication mechanisms, including multi-factor authentication for accessing sensitive systems and data. It applies to all users accessing the organization’s network, systems, and applications. Authentication ensures that only authorized individuals can gain access, reducing the risk of data breaches. |
| Authorization | Authorization defines what actions and resources a user or system is allowed to access after successful authentication. The policy specifies role-based access control and least privilege principles. It applies to all users and systems and ensures that users only have access to the data and resources necessary for their roles. Authorization policies help prevent unauthorized access and data misuse. |
| Accounting | Accounting, also known as auditing or logging, involves tracking and recording activities related to user logins, changes to the database, addition of new users, user access levels, and files accessed by users. The policy mandates the continuous monitoring and logging of these activities. It applies to all systems and sensitive data. Accounting assists in detecting and investigating security incidents, ensuring compliance, and providing a record of who accessed what and when. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

Works Cited

*Sei External Wiki Home*. SEI External Wiki Home - Homepage - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/