**Green Pace Developer: Security Policy**



# 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 | Verifying users and data is the first step at reducing software vulnerabilities from external sources. |
| 1. Heed Compiler Warnings | Use the highest warning level available for the specific compiler being used and ensure the analysis tools eliminate any security weaknesses. |
| 1. Architect and Design for Security Policies | The main concerns should be based around the design of software for security. This could result in changes to the way the software operates but can create multiple defensive layers that can reduce attacks. |
| 1. Keep It Simple | Keeping the code simple will make it much less confusing and easier to find errors, later down the line if another developer is tasked at working on the security tools, and the more complicated it becomes, the harder it will be to use or fix/debug. |
| 1. Default Deny | If access is denied by default, it will be easier to restrict permission and deny those who do not or should not have access. |
| 1. Adhere to the Principle of Least Privilege | The least number of privileges necessary to complete each task should be used. Using the least amount of time will reduce the attack opportunity to exploit elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Data should be passed through and sanitized to avoid attacks using unintentional commands or injection attacks. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| 1. Practice Defense in Depth | Have multiple layers of defense and more than one plan, an additional layer of defense can stop an attack or protect the data from a vulnerability that may be bypassed with a single layer of security. |
| 1. Use Effective Quality Assurance Techniques | Quality Assurance teams need to be used to go over the code and ensure there are no vulnerabilities or flaws, and these should be external teams in order to get a fresh perspective on the security program. |
| 1. Adopt a Secure Coding Standard | Follow along with the best practices for your secure coding standard on whichever platform you plan to implement it on, and don’t deviate. |

### 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** | **EXP59-CPP** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Use offset() on valid types and members. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example, a type that is not a standard-layout class is passed to the offsetof() macro, resulting in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <cstddef>    struct D {    virtual void f() {}    int i;  };    void f() {    size\_t off = offsetof(D, i);    // ...  } |

| **Compliant Code** |
| --- |
| It is not possible to determine the offset to i within D because D is not a standard-layout class. However, it is possible to make a standard-layout class within D if this functionality is critical to the application, as demonstrated by this compliant solution. |
| #include <cstddef>    struct D {    virtual void f() {}    struct InnerStandardLayout {      int i;    } inner;  };    void f() {    size\_t off = offsetof(D::InnerStandardLayout, i);    // ...  } |

| **Principles(s):** Validate Input Data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-EXP59 | N/A |
| Clang | 4.0 | -Winvalid-offsetof | Emits an error diagnostic on invalid member designators and emits a warning diagnostic on invalid types. |
| GCC | 4.9 | -Winvalid-offsetof | Emits an error diagnostic on invalid member designators and emits a warning diagnostic on invalid types. |
| Helix QAC | 2021.3 | C++3915, C++3916 | N/A |

#### Coding Standard 2

| **Coding Standard** | **Label** | **CTR58-CPP** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Predicate function objects should not be mutable. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to remove the third item in a container using a predicate that returns true only on its third invocation. |
| #include <algorithm>  #include <functional>  #include <iostream>  #include <iterator>  #include <vector>    class MutablePredicate : public std::unary\_function<int, bool> {    size\_t timesCalled;  public:    MutablePredicate() : timesCalled(0) {}      bool operator()(const int &) {      return ++timesCalled == 3;    }  };    template <typename Iter>  void print\_container(Iter b, Iter e) {    std::cout << "Contains: ";    std::copy(b, e, std::ostream\_iterator<decltype(\*b)>(std::cout, " "));    std::cout << std::endl;  }    void f() {    std::vector<int> v{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};    print\_container(v.begin(), v.end());      v.erase(std::remove\_if(v.begin(), v.end(), MutablePredicate()), v.end());    print\_container(v.begin(), v.end());  } |

| **Compliant Code** |
| --- |
| This compliant solution wraps the predicate in a std::reference\_wrapper<T> object, ensuring that copies of the wrapper object all refer to the same underlying predicate object. |
| #include <algorithm>  #include <functional>  #include <iostream>  #include <iterator>  #include <vector>    class MutablePredicate : public std::unary\_function<int, bool> {    size\_t timesCalled;  public:    MutablePredicate() : timesCalled(0) {}      bool operator()(const int &) {      return ++timesCalled == 3;    }  };    template <typename Iter>  void print\_container(Iter b, Iter e) {    std::cout << "Contains: ";    std::copy(b, e, std::ostream\_iterator<decltype(\*b)>(std::cout, " "));    std::cout << std::endl;  }    void f() {    std::vector<int> v{0, 1, 2, 3, 4, 5, 6, 7, 8, 9};    print\_container(v.begin(), v.end());      MutablePredicate mp;    v.erase(std::remove\_if(v.begin(), v.end(), std::ref(mp)), v.end());    print\_container(v.begin(), v.end());  } |

| **Principles(s):** Default Deny. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.3 | C++3225, C++3226, C++3227, C++3228, C++3229, C++3230, C++3231, C++3232, C++3233, C++3234 | N/A |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-CTR58-a | Make predicates const pure functions |
| PRQA QA-C++ | 4.4 | 3225, 3226, 3227, 3228, 3229, 3230, 3231, 3232, 3233, 3234 | N/A |
| Klocwork | 2021.4 | CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE | N/A |

#### Coding Standard 3

| **Coding Standard** | **Label** | **STR52-CPP** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Use valid references, pointers, and iterators to reference elements of a basic\_string. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example copies input into a std::string, replacing semicolon (;) characters with spaces. This example is noncompliant because the iterator loc is invalidated after the first call to insert(). The behavior of subsequent calls to insert() is undefined. |
| #include <string>    void f(const std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();    for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the value of the iterator loc is updated as a result of each call to insert() so that the invalidated iterator is never accessed. The updated iterator is then incremented at the end of the loop. |
| #include <string>    void f(const std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();    for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      loc = email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

| **Principles(s):** Keep it Simple. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.3 | C++4746, C++4747, C++4748, C++4749 | N/A |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |
| Astrée | 21.10 | assert\_failure | N/A |
| N/A | N/A | N/A | N/A |

#### Coding Standard 4

| **Coding Standard** | **Label** | **IDS00-J** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-J] | 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](https://wiki.sei.cmu.edu/confluence/display/java/Rule+BB.+Glossary#RuleBB.Glossary-sanitize) 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      }    }  } |

| **Principles(s):** Implement a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| Coverity | 2021.01 | SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Findbugs | 1.3.9 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| CodeSonar | 6.2 | JAVA.IO.INJ.SQL | SQL Injection - Java |

#### Coding Standard 5

| **Coding Standard** | **Label** | **MEM50-CPP** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory. |

| **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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-vulnerability) can be [exploited](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-exploit) to run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #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;  } |

| **Principles(s):** Practice Defense in Depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.3 | C++3225, C++3226, C++3227, C++3228, C++3229, C++4632 | N/A |
| Compass/ROSE | N/A | N/A | N/A |
| LDRA tool suite | 9.7.2 | 45 D | Partially implemented |
| Coverity | 2021.01 | CHECKED\_RETURN | Finds inconsistencies in how function call returned values are handled |

#### Coding Standard 6

| **Coding Standard** | **Label** | **DCL03-C** |
| --- | --- | --- |
| **Assertions** | [STD-006-C] | Use static assertions to test constant expressions to identify vulnerabilities using preprocessor conditional statements to avoid the assert being skipped over in runtime execution. |

| **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 |

| **Principles(s):** Use Quality Assurance Techniques. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.3 | C++2668, C++2761, C++2762, C++2763, C++2766, C++2767, C++2768 | N/A |
| Astrée | 21.10 | Invalid\_pointer\_subtraction invalid\_pointer\_comparison | N/A |
| LDRA tool suite | 9.7.2 | 70 S, 87 S, 437 S, 438 S | Enhanced Enforcement |
| Clang | 4.0 | misc-static-assert | Checked by clang-tidy |

#### Coding Standard 7

| **Coding Standard** | **Label** | **ERR51-CPP** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code 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** |
| --- |
| In this compliant solution, 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    }  } |

| **Principles(s):** Default Deny |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.3 | C++2045, C++2047, C++4032, C++4631 | N/A |
| Astrée | 21.10 | Destructor\_without\_noexcept delte\_without\_noexcept | Fully checked |
| LDRA tool suite | 9.7.2 | 453 S | Partially Implemented |
| Axivion Bauhaus Suite | 7.2.0 | CertC++ERR51 | N/A |

#### Coding Standard 8

| **Coding Standard** | **Label** | **CTR52-CPP** |
| --- | --- | --- |
| **Overflows** | [STD-008-CPP] | Guarantee that library functions do not overflow. |

| **Noncompliant Code** |
| --- |
| STL containers can be subject to the same vulnerabilities as array data types. The std::copy() algorithm provides no inherent bounds checking and can lead to a buffer overflow. In this noncompliant code example, a vector of integers is copied from src to dest using std::copy(). Because std::copy() does nothing to expand the dest vector, the program will overflow the buffer on copying the first element. |
| #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 proper way to use std::copy() is to ensure the destination container can hold all the elements being copied to it. This compliant 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());    // ...  } |

| **Principles(s):** Avoid Buffer Overflow. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P4 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.3 | C++3802 | N/A |
| Astrée | 21.10 | overflow\_upon\_dereference | N/A |
| PRQA QA-C++ | 4.4 | 3802 | N/A |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-CTR52-a  CERT\_CPP-CTR52-b | Do not use an iterator range that is not really a range  Do not compile iterators from different containers |

#### Coding Standard 9

| **Coding Standard** | **Label** | **FIO01-CPP** |
| --- | --- | --- |
| **Access Permissions** | [STD-009-CPP] | Create files with appropriate access permissions. |

| **Noncompliant Code** |
| --- |
| The constructors for FileOutputStream and FileWriter do not allow the programmer to explicitly specify file access permissions. In this noncompliant code example, the access permissions of any file created are implementation-defined and may not prevent unauthorized access: |
| Writer out = new FileWriter("file"); |

| **Compliant Code** |
| --- |
| The  I/O facility [java.nio](http://docs.oracle.com/javase/7/docs/api/java/nio/package-summary.html) provides classes for managing file access permissions. Additionally, many of the methods and constructors that create files accept an argument allowing the program to specify the initial file permissions.  The Files.newByteChannel() method allows a file to be created with specific permissions. This method is platform-independent, but the actual permissions are platform-specific. This compliant solution defines sufficiently restrictive permissions for POSIX platforms: |
| Path file = new File("file").toPath();    // Throw exception rather than overwrite existing file  Set<OpenOption> options = new HashSet<OpenOption>();  options.add(StandardOpenOption.CREATE\_NEW);  options.add(StandardOpenOption.APPEND);    // File permissions should be such that only user may read/write file  Set<PosixFilePermission> perms =      PosixFilePermissions.fromString("rw-------");  FileAttribute<Set<PosixFilePermission>> attr =      PosixFilePermissions.asFileAttribute(perms);    try (SeekableByteChannel sbc =           Files.newByteChannel(file, options, attr)) {    // Write data  }; |

| **Principles(s):** Implement a secure coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | High | P4 | L4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.3 | C++3112 | N/A |
| LDRA tool suite | 9.7.2 | 169 S, 554 S | Enhanced Enforcement |
| Parasoft Insure++ | N/A | N/A | Runtime detection |
| RuleChecker | 20.10 | Delete-with-incomplete-type | N/A |

#### Coding Standard 10

| **Coding Standard** | **Label** | **FIO51-CPP** |
| --- | --- | --- |
| **Close Files** | [STD-010-CPP] | Close files when they are no longer needed. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    file.close();    if (file.fail()) {      // Handle error    }    std::terminate();  } |

| **Principles(s):** Use effective quality assurance techniques. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.3 | C++4786, C++4787, C++4788 | N/A |
| Klocwork | 2021.4 | RH.LEAK | N/A |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-FIO51-a | Ensures resources are freed |
| CodeSonar | 6.2 | ALLOC.LEAK | Leak |

### 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.

The DevOps process shown here for Green Pace can be automated at the Transition and health check phase of the production cycle. This step in the policy is where recently verified and tested settings are prepared for deployment and the specific security settings can be pre setup and automated by pushing updates with those rules in place. In addition, if a penetration testing lab is used and was automated to repetitively test these new guidelines, and it would not have to be done manually and have a chance of problems with human error. Lastly, the system would begin by closing necessary ports, blocking IPs, and turning off services when needed, and that would make the job of the response team easier to be able to monitor the system for penetration.

### 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 | Medium | Unlikely | Medium | 4 | 3 |
| STD-002-CPP | Low | Likely | High | 3 | 3 |
| STD-003-CPP | High | Probable | High | 6 | 2 |
| STD-004-J | High | Probable | Medium | 12 | 1 |
| STD-005-CPP | High | Likely | Medium | 18 | 1 |
| STD-006-C | Medium | Probable | Medium | 8 | 2 |
| STD-007-CPP | Low | Likely | Medium | 6 | 2 |
| STD-008-CPP | High | Probable | High | 4 | 2 |
| STD-009-CPP | Medium | Probable | High | 4 | 4 |
| STD-010-CPP | Medium | Unlikely | Medium | 4 | 3 |

### 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 | This involves encrypting data that is stored on a disk or other drive. In order for someone to gain access to an encrypted drive and they do not have the keys, they won’t be able to access it. |
| Encryption at flight | This is the process of encrypting data that is transmitted, this would apply to all data between servers and devices that could be compromised. |
| Encryption in use | This is securing data by ensuring it’s never left vulnerable. This would include using permission-based roles in order to mitigate attacks on data in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authenticating users via usernames, passwords and other mechanisms applies when utilizing standards of default deny and permission-based coding. It can also keep logs of what user have done with the credentials. |
| Authorization | This follows authentication and is a second layer of authorizing a user for a particular set of tasks and access to relevant data. This ensures that something is not modified by someone that should not have access to it, and it also applies to useful coding standard practice. |
| Accounting | This keeps track of all the resources used by the users in the system. It can help keep track of things like, the length of their session or what data was sent and received. This is important for analysis and keeping a secure log of activity. |

**\***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 | 02/11/2022 | Information Input | Andrew Wepplo | N/A |
| [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 |