**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 | “Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability). Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files [Seacord 05].” (Seacord & Schiela, 2018) |
| 1. Heed Compiler Warnings | “Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code [[C MSC00-A](https://wiki.sei.cmu.edu/confluence/display/c/MSC00-C.+Compile+cleanly+at+high+warning+levels), [C++ MSC00-A](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046361)]. Use static and dynamic analysis tools to detect and eliminate additional security flaws.” (Seacord & Schiela, 2018) |
| 1. Architect and Design for Security Policies | “Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set.” (Seacord & Schiela, 2018) |
| 1. Keep It Simple | “Keep the design as simple and small as possible [Saltzer 74, Saltzer 75]. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex.” (Seacord & Schiela, 2018) |
| 1. Default Deny | “Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted [Saltzer 74, Saltzer 75].” (Seacord & Schiela, 2018) |
| 1. Adhere to the Principle of Least Privilege | “Every process should execute with the the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges [Saltzer 74, Saltzer 75].” (Seacord & Schiela, 2018) |
| 1. Sanitize Data Sent to Other Systems | “Sanitize all data passed to complex subsystems [[C STR02-A](https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems)] such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem.” (Seacord & Schiela, 2018) |
| 1. Practice Defense in Depth | “Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a [security flaw](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-securityflaw) from becoming an exploitable vulnerability and/or limit the consequences of a successful [exploit](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-exploit). For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment [Seacord 05].” (Seacord & Schiela, 2018) |
| 1. Use Effective Quality Assurance Techniques | “Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions [Seacord 05].” (Seacord & Schiela, 2018) |
| 1. Adopt a Secure Coding Standard | “Develop and/or apply a secure coding standard for your target development language and platform.” (Seacord & Schiela, 2018) |

## 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** |
| --- |
| In this 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);    // ...  } |

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

|  |
| --- |
| **Principles(s):** **Validate input.** |

**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 | 3.9 | -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 | 2021.2 | **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());  } |

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

|  |
| --- |
| **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 | 2021.2 | **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 |
| N/A | N/A | N/A | 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 : ' ');    }  } |

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

|  |
| --- |
| **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 | 2021.2 | **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 |
| N/A | N/A | N/A | N/A |
| N/A | N/A | N/A | N/A |

### Coding Standard 4

| **Coding Standard** | **Label** | **IDS00-J** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | 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      }    }  } |

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

|  |
| --- |
| **Principles(s):** **Adopt 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 (see Chapter 8) |
| CodeSonar | 6.1p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| 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** | **MEM52-CPP** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| In this noncompliant code 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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination) 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.**

|  |
| --- |
| **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** |
| --- | --- | --- | --- |
| Compass/ROSE | N/A | N/A | N/A |
| Coverity | 7.5 | **CHECKED\_RETURN** | Finds inconsistencies in how function call return values are handled |
| Helix QAC | 2021.2 | **C++3225, C++3226, C++3227, C++3228, C++3229, C++4632** | N/A |
| LDRA tool suite | 9.7.1 | 45 D | Partially Implemented |

### Coding Standard 6

| **Coding Standard** | **Label** | **CTR54-CPP** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Do not subtract iterators that do not refer to the same container. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to determine whether the pointer test is within the range [r, r + n]. However, when test does not point within the given range, as in this example, the subtraction produces undefined behavior. |
| #include <cstddef>  #include <iostream>    template <typename Ty>  bool in\_range(const Ty \*test, const Ty \*r, size\_t n) {    return 0 < (test - r) && (test - r) < (std::ptrdiff\_t)n;  }    void f() {    double foo[10];    double \*x = &foo[0];    double bar;    std::cout << std::boolalpha << in\_range(&bar, x, 10);  } |

| **Compliant Code** |
| --- |
| This compliant solution demonstrates a fully portable, but likely inefficient, implementation of in\_range() that compares test against each possible address in the range [r, n]. A compliant solution that is both efficient and fully portable is currently unknown. |
| #include <iostream>    template <typename Ty>  bool in\_range(const Ty \*test, const Ty \*r, size\_t n) {    auto \*cur = reinterpret\_cast<const unsigned char \*>(r);    auto \*end = reinterpret\_cast<const unsigned char \*>(r + n);    auto \*testPtr = reinterpret\_cast<const unsigned char \*>(test);      for (; cur != end; ++cur) {      if (cur == testPtr) {        return true;      }    }    return false;  }    void f() {    double foo[10];    double \*x = &foo[0];    double bar;    std::cout << std::boolalpha << in\_range(&bar, x, 10);  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **invalid\_pointer\_subtraction invalid\_pointer\_comparison** | N/A |
| Helix QAC | 2021.2 | **C++2668, C++2761, C++2762, C++2763, C++2766, C++2767, C++2768** | N/A |
| LDRA tool suite | 9.7.1 | **70 S, 87 S, 437 S, 438 S** | Enhanced Enforcement |
| PRQA QA-C++ | 4.4 | **2668, 2761, 2762, 2763, 2766, 2767, 2768** | Enforced by QA-CPP |

### Coding Standard 7

| **Coding Standard** | **Label** | **DCL57-CPP** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not let exceptions escape from destructors or deallocation functions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the class destructor does not meet the implicit noexcept guarantee because it may throw an exception even if it was called as the result of an exception being thrown. Consequently, it is declared as noexcept(false) but still can trigger [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <stdexcept>    class S {    bool has\_error() const;    public:    ~S() noexcept(false) {      // Normal processing      if (has\_error()) {        throw std::logic\_error("Something bad");      }    }  }; |

| **Compliant Code** |
| --- |
| A destructor should perform the same way whether or not there is an active exception. Typically, this means that it should invoke only operations that do not throw exceptions, or it should handle all exceptions and not rethrow them (even implicitly). This compliant solution differs from the previous noncompliant code example by having an explicit return statement in the SomeClass destructor. This statement prevents control from reaching the end of the exception handler. Consequently, this handler will catch the exception thrown by Bad::~Bad() when bad\_member is destroyed. It will also catch any exceptions thrown within the compound statement of the function-try-block, but the SomeClass destructor will not terminate by throwing an exception. |
| class SomeClass {    Bad bad\_member;  public:    ~SomeClass()    try {      // ...    } catch(...) {      // Catch exceptions thrown from noncompliant destructors of      // member objects or base class subobjects.        // NOTE: Flowing off the end of a destructor function-try-block causes      // the caught exception to be implicitly rethrown, but an explicit      // return statement will prevent that from happening.      return;    }  }; |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **destructor-without-noexcept delete-without-noexcept** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL57** | N/A |
| Helix QAC | 2021.2 | **C++2045, C++2047, C++4032, C++4631** | N/A |
| LDRA tool suite | 9.7.1 | 453 S | Partially Implemented |

### Coding Standard 8

| **Coding Standard** | **Label** | **OOP51-CPP** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Do not slice derived objects |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an object of the derived Manager type is passed by value to a function accepting a base Employee type. Consequently, the Manager objects are sliced, resulting in information loss and unexpected behavior when the print() function is called. |
| #include <iostream>  #include <string>    class Employee {    std::string name;    protected:    virtual void print(std::ostream &os) const {      os << "Employee: " << get\_name() << std::endl;    }    public:    Employee(const std::string &name) : name(name) {}    const std::string &get\_name() const { return name; }    friend std::ostream &operator<<(std::ostream &os, const Employee &e) {      e.print(os);      return os;    }  };    class Manager : public Employee {    Employee assistant;    protected:    void print(std::ostream &os) const override {      os << "Manager: " << get\_name() << std::endl;      os << "Assistant: " << std::endl << "\t" << get\_assistant() << std::endl;    }    public:    Manager(const std::string &name, const Employee &assistant) : Employee(name), assistant(assistant) {}    const Employee &get\_assistant() const { return assistant; }  };    void f(Employee e) {    std::cout << e;  }    int main() {    Employee coder("Joe Smith");    Employee typist("Bill Jones");    Manager designer("Jane Doe", typist);      f(coder);    f(typist);    f(designer);  } |

| **Compliant Code** |
| --- |
| Using the same class definitions as the noncompliant code example, this compliant solution modifies the definition of f() to require raw pointers to the object, removing the slicing problem. |
| // Remainder of code unchanged...    void f(const Employee \*e) {    if (e) {      std::cout << \*e;    }  }    int main() {    Employee coder("Joe Smith");    Employee typist("Bill Jones");    Manager designer("Jane Doe", typist);      f(&coder);    f(&typist);    f(&designer);  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | **C++3072** | N/A |
| Parasoft C/C++test | 2021.1 | **CERT\_CPP-OOP51-a** | Avoid slicing function arguments / return value |
| Polyspace Bug Finder | R2021a | [CERT C++: OOP51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcoop51cpp.html) | Checks for object slicing (rule partially covered) |
| PRQA QA-C++ | 4.4 | 3072 | N/A |

### Coding Standard 9

| **Coding Standard** | **Label** | **CTR52-CPP** |
| --- | --- | --- |
| [Student Choice] | [STD-009-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());    // ...  } |

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

|  |
| --- |
| **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** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **invalid\_pointer\_dereference** | N/A |
| 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 | 2021.2 | **C++3526, C++3527, C++3528, C++3529, C++3531, C++3532, C++3533, C++3534** | N/A |
| Parasoft C/C++test | 2021.1 | **CERT\_CPP-CTR52-a** | Do not pass empty container iterators to std algorithms as destinations |

### Coding Standard 10

| **Coding Standard** | **Label** | **FIO51-CPP** |
| --- | --- | --- |
| [Student Choice] | [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();  } |

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

|  |
| --- |
| **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** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | **ALLOC.LEAK** | Leak |
| Helix QAC | 2021.2 | **C++4786, C++4787, C++4788** | N/A |
| Klocwork | 2021.1 | RH.LEAK | N/A |
| Parasoft C/C++test | 2021.1 | **CERT\_CPP-FIO51-a** | Ensures resources are freed. |

## 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 for Green Pace can be automated for enforcement at the Transition and Health check phase of the cycle. This is the part of the process where recently verified and tested settings are deployed. The deployment of these security settings can be automated by pushing updates with a set of pre-configured rules and standards to all live systems. This will alleviate human error that can happen from configuring these rules and standards manually. Normally penetration testing is done manually; however, a penetration testing lab could be automated to constantly test the newly released settings. If this automated system can gain unauthorized access to the new code, it will trigger an event immediately. Finally, the system would begin by closing necessary ports, blocking IPs, and turning off services as needed. At this point, the response team would be able to monitor and stabilize the system.

## 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-CPP | High | Probable | Medium | 12 | 1 |
| STD-005-CPP | High | Likely | Medium | 18 | 1 |
| STD-006-CPP | Medium | Probable | Medium | 8 | 2 |
| STD-007-CPP | Low | Likely | Medium | 6 | 2 |
| STD-008-CPP | Low | Probable | Medium | 4 | 3 |
| STD-009-CPP | High | Likely | Medium | 18 | 1 |
| 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 is designed to prevent anyone attempting to attack the system to access the information directly from a hard drive by ensuring that the data on the drive is encrypted. In order to gain access they would need to beat the encryption first in order to read the data. This can be important in the event that your physical devices get stolen or misplaced. |
| Encryption at flight | This is the process of encrypting information that is being transmitted. Sometimes the information may not be encrypted while its at rest, but will be encrypted during transmission in order to protect it. This can be a good way of protecting any information from falling into the wrong hands so to speak, though it is considerably more vulnerable than encryption in rest. |
| Encryption in use | This is the process of ensuring that all of the information is always secured and encrypted and never left vulnerable. This is the most secure of the practices and should at best always be used if possible. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is generally the first step in a Triple-A framework. It helps to ensure that only those that are supposed to have access are let in. It can also keep a log of what a user does under the credentials that have been authenticated. |
| Authorization | This is the step that determines what the user is allowed to do in the system. Can they make changes or just read documents? Is there a certain time that they should only ever be logged on? This can be used to ensure that something is not accidentally altered by someone that should not have access to it. |
| 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. It can help with billing and keeping track of trends. |

**\***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 | 08/09/2021 | Information inputted | Angelica Hayes | [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 |

# References

Seacord, R., & Schiela, R. (2018, May 2). *Top 10 Secure Coding Practices - CERT Secure Coding - Confluence*. Top 10 Secure Coding Practices - CERT Secure Coding - Confluence. <https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices>

Schiela, R. (2020, May 29). *SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence*. SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence. https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682