**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 | Data originating from untrusted sources can result in incorrect input. By implementing rigorous input validation techniques, we can effectively prevent a significant portion of software vulnerability exploits. |
| 1. Heed Compiler Warnings | Compiler warnings provide valuable insights into potential issues within your code. Addressing and resolving these warnings enhances the security of your code compared to leaving them unattended. |
| 1. Architect and Design for Security Policies | When developing software, prioritize security in both design and implementation. Ensure the use of appropriate variables for handling incoming and outgoing data. Additionally, consider establishing subsystems that communicate with each other based on the required level of privileges. |
| 1. Keep It Simple | As the complexity and intricacy of software design increases, so does the likelihood of vulnerabilities. Simplifying the design facilitates the implementation of security modifications and simplifies security maintenance tasks within the system.  Top of Form |
| 1. Default Deny | Set up the software to automatically limit access to information unless certain criteria are met.  Top of Form  Top of Form |
| 1. Adhere to the Principle of Least Privilege | Grant each level of permission access only to the necessary information. If an individual requires access to a specific subset of data, provide access solely to that subset. |
| 1. Sanitize Data Sent to Other Systems | If a system feature remains unused, it should be eliminated. Attackers may exploit unused functionalities to extract additional information. By removing such functions and ensuring the sanitization of transmitted data, only essential elements remain in use or available.  Top of Form |
| 1. Practice Defense in Depth | Implementing multiple layers of security with overlapping functionalities increases the difficulty for hackers to obtain information. Additionally, it ensures the system remains secure even if one layer of defense is breached. |
| 1. Use Effective Quality Assurance Techniques | Employing security testing and other quality assurance (QA) methods aids in pinpointing vulnerabilities and issues prior to the deployment of changes or programs. This proactive approach helps mitigate the risk of data leaks and successful attacks in the future. |
| 1. Adopt a Secure Coding Standard | Establish and adhere to a secure coding standard to ensure consistency and effectiveness. This standardized format facilitates swift and straightforward implementation of security changes and upgrades across all aspects of the system. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-C | Implement abstract data types using opaque data types |

| **Noncompliant Code** |
| --- |
| The implementation of the string\_mx type is fully visible to the user of the data type after including the string\_m.h file. Programmers are consequently more likely to directly manipulate the fields within the structure, violating the software engineering principles of information hiding and data encapsulation and increasing the probability of developing incorrect or nonportable code. |
| **struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;    unsigned **char** strtype;  **char** \*cstr;  };    **typedef** **struct** string\_mx string\_mx;    /\* Function declarations \*/  **extern** errno\_t strcpy\_m(string\_mx \*s1, **const** string\_mx \*s2);  **extern** errno\_t strcat\_m(string\_mx \*s1, **const** string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| This compliant solution reimplements the string\_mx type as a private type, hiding the implementation of the data type from the user of the managed string library. To accomplish this, the developer of the private data type creates two header files: an external string\_m.h header file that is included by the user of the data type and an internal file that is included only in files that implement the managed string abstract data type. |
| struct string\_mx;  typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/ |

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

| **Principles(s):** “Validate Input data”: We want to make sure that we are implementing abstract data types using opaque data types. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhuas Suite | 7.2.0 | CertC-DCL12 |  |
| LDRA tool suite | 97.1 | 104 D | Partially implemented |
| Polyspace Bug Finder | R20223b | Cert C: Red. DCL12-c | Checks for structure or union object implementation visible in file where pointer to this object is not dereferenced (rule partially covered) |
| Parasoft C/C++ test | 2023.1 | CERT\_C-DCL12-a | If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-C | Ensure that integer conversions do not result in lost or misinterpreted data |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type. This noncompliant code example results in a truncation error on most implementations |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Validate ranges when converting from an unsigned type to a signed type. This compliant solution can be used to convert a value of unsigned long int type to a value of signed char type |
| #include <limits.h>    void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  if (u\_a <= SCHAR\_MAX) {  sc = (signed char)u\_a; /\* Cast eliminates warning \*/  }  else {  /\* Handle error \*/  }  } |

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

| **Principles(s):** “Validate Input Data”: We want to make sure that after an int conversion that all the data is intact. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| LDRA tool suite | 9.7.1 | 93 S, 433 S, 434 S | Partially implemented |
| TrustinSoft Analyzer | 1.38 | signed\_downcast | Exhaustively verified |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-C | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| This noncompliant code example converts the string token stored in the buff to a signed integer value using the atoi() function |
| #include <stdlib.h>    void func(const char \*buff) {  int si;    if (buff) {  si = atoi(buff);  } else {  /\* Handle error \*/  }  } |

| **Compliant Code** |
| --- |
| This compliant solution uses strtol() to convert a string token to an integer and ensures that the value is in the range of int |
| #include <errno.h>  #include <limits.h>  #include <stdlib.h>  #include <stdio.h>    void func(const char \*buff) {  char \*end;  int si;    errno = 0;    const long sl = strtol(buff, &end, 10);      if (end == buff) {  (void) fprintf(stderr, "%s: not a decimal number\n", buff);  } else if ('\0' != \*end) {  (void) fprintf(stderr, "%s: extra characters at end of input: %s\n", buff, end);  } else if ((LONG\_MIN == sl || LONG\_MAX == sl) && ERANGE == errno) {  (void) fprintf(stderr, "%s out of range of type long\n", buff);  } else if (sl > INT\_MAX) {  (void) fprintf(stderr, "%ld greater than INT\_MAX\n", sl);  } else if (sl < INT\_MIN) {  (void) fprintf(stderr, "%ld less than INT\_MIN\n", sl);  } else {  si = (int)sl;    /\* Process si \*/  }  } |

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

| **Principles(s):** “Adopt a Secure Coding Standard”: We want to make sure that we catch errors. We need proper standards to set up measures to do this. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhuas Suite | 7.2.0 | CertC++-ERR62 |  |
| Clang | 3.9 | cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| CodeSonar | 8.1p0 | BADFUNC.ATOF  BADFUNC.ATOI  BADFUNC.ATOL  BADFUNC.ATOLL | Use of atof  Use of atoi  Use of atol  Use of atoll |
| 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** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-J | Prevent SQL injection |

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

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

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

| **Principles(s):** “Architect and Design for Security Policies”: This applies because we are making sure that an SQL injection does not occur.  “Sanitize Data Sent to Other Systems” this applies because we want to make sure that data sent to another system will not compromise the data or the system. This could cause an unexpected outcome. |
| --- |

**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 | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Parasoft Jtest | 2023.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Prevent data races when accessing bit-fields from multiple threads |

| **Noncompliant Code** |
| --- |
| Adjacent bit-fields may be stored in a single memory location. Consequently, modifying adjacent bit-fields in different threads is undefined behavior, as shown in this noncompliant code example |
| struct 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” this applies because data races will cause a big mess by having two or more sources accessing the same memory location.  “Adopt a Secure Coding Standard” this applies because with a good coding standard this can be avoided. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | read\_write\_data\_race  write\_write\_data\_race | Supported |
| Axivion Bauhaus Suite | 7.2.0 | CERT++-CON52 |  |
| CodeSonar | 8.1p0 | CONCURRENCY.DATARACE | Data race |
| Coverity | 6.5 | Race\_Condition | Fully implemented |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-C | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that 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):** “Adopt a Secure Coding Standard” this applies because using the correct assertion is about how well the programmer knows about them. With a good standard they will know what to do. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CERTC-DCL03 |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 8.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Exception objects must be nothrow copy constructible |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an exception of type S is thrown in f(). However, because S has a std::string data member, and the copy constructor for std::string is not declared noexcept, the implicitly-defined copy constructor for S is also not declared to be noexcept. In low-memory situations, the copy constructor for std::string may be unable to allocate sufficient memory to complete the copy operation, resulting in a std::bad\_alloc exception being thrown |
| #include <exception>  #include <string>    class S : public std::exception {  std::string m;  public:  S(const char \*msg) : m(msg) {}    const char \*what() const noexcept override {  return m.c\_str();  }  };    void g() {  // If some condition doesn't hold...  throw S("Condition did not hold");  }    void f() {  try {  g();  } catch (S &s) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| This compliant solution assumes that the type of the exception object can inherit from std::runtime\_error, or that type can be used directly. Unlike std::string, a std::runtime\_error object is required to correctly handle an arbitrary-length error message that is exception safe and guarantees the copy constructor will not throw [ ISO/IEC 14882-2014 ] |
| #include <stdexcept>  #include <type\_traits>    struct S : std::runtime\_error {  S(const char \*msg) : std::runtime\_error(msg) {}  };    static\_assert(std::is\_nothrow\_copy\_constructible<S>::value,  "S must be nothrow copy constructible");    void g() {  // If some condition doesn't hold...  throw S("Condition did not hold");  }    void f() {  try {  g();  } catch (S &s) {  // Handle error  }  } |

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

| **Principles(s):** “Adopt a Secure Coding Standard” this applies because you have to make sure that you are following the standard. Otherwise, there will be problems with the program. A good standard will prevent this from happening. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | cert-err60-cpp | Checked by clang-tidy |
| Helix QAC | 2024.1 | C++3508 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-ERR60-a  CERT\_CPP-ERR60-b | Exception objects must be nothrow copy constructible  An explicitly declared copy constructor for a class that inherits from 'std::exception' should have a non-throwing exception specification |
| Polyspace Bug Finder | R2023b | CERT C++: ERR60-CPP | Checks for throwing exception object in copy constructor (rule fully covered) |

#### Coding Standard 8

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

| **Noncompliant Code** |
| --- |
| This noncompliant code example appends data to the end of a file and then reads from the same file. However, because there is no intervening positioning call between the formatted output and input calls, the behavior is undefined |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output some data";  std::string str;  file >> str;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::basic\_istream<T>::seekg() function is called between the output and input, eliminating the undefined behavior |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output some data";    std::string str;  file.seekg(0, std::ios::beg);  file >> str;  } |

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

| **Principles(s):** “Use Effective Quality Assurance Techniques” this applies because using good testing during QA will make sure this is caught and fixed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-FIO50 |  |
| CodeSonar | 8.1p0 | IO.IOWOP  IO.OIWOP | Input After Output Without Positioning  Output After Input Without Positioning |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polysapce Bug Finder | R2023b | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Objected oriented programming | STD-009-CPP | Do not invoke virtual functions from constructors or destructors |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the base class attempts to seize and release an object's resources through calls to virtual functions from the constructor and destructor. However, the B::B() constructor calls B::seize() rather than D::seize(). Likewise, the B::~B() destructor calls B::release() rather than D::release() |
| struct B {  B() { seize(); }  virtual ~B() { release(); }    protected:  virtual void seize();  virtual void release();  };    struct D : B {  virtual ~D() = default;    protected:  void seize() override {  B::seize();  // Get derived resources...  }    void release() override {  // Release derived resources...  B::release();  }  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the constructors and destructors call a nonvirtual, private member function (suffixed with mine) instead of calling a virtual function. The result is that each class is responsible for seizing and releasing its own resources |
| class B {  void seize\_mine();  void release\_mine();    public:  B() { seize\_mine(); }  virtual ~B() { release\_mine(); }    protected:  virtual void seize() { seize\_mine(); }  virtual void release() { release\_mine(); }  };    class D : public B {  void seize\_mine();  void release\_mine();    public:  D() { seize\_mine(); }  virtual ~D() { release\_mine(); }    protected:  void seize() override {  B::seize();  seize\_mine();  }    void release() override {  release\_mine();  B::release();  }  }; |

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

| **Principles(s):** “Use Effective Quality Assurance Techniques” this applies because during QA this should be caught and Fixed.  “Adopt a Secure Coding Standard” this applies because with a good standard this should not occur. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | virtual-call-in-constructor  invalid\_function\_pointer | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP50 |  |
| Clang | 3.9 | clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clang-tidy |
| CodeSonar | 8.1p0 | LANG.STRUCT.VCALL\_IN\_CTOR  LANG.STRUCT.VCALL\_IN\_DTOR | Virtual Call in Constructor  Virtual Call in Destructor |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Loops | STD-010-J | Do not use an empty infinite loop |

| **Noncompliant Code** |
| --- |
| This noncompliant code example implements an idle task that continuously executes a loop without executing any instructions within the loop. An optimizing compiler or JIT could remove the while loop in this example |
| public int nop() {  while (true) {}  } |

| **Compliant Code** |
| --- |
| This compliant solution avoids use of a meaningless infinite loop by invoking Thread.sleep() within the while loop. The loop body contains semantically meaningful operations and consequently cannot be optimized away |
| public final int DURATION=10000; // In milliseconds    public void nop() throws InterruptedException {  while (true) {  // Useful operations  Thread.sleep(DURATION);  }  } |

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

| **Principles(s):** “Adopt a Secure Coding Standard” this applies because with a good standard there should not be an empty infinite loop. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Jtest | 2023.1 | CERT.MSC01.EB | Avoid control statements with empty bodies |
| SonarQube | 3.10 | S2189 |  |

### 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.

Automating the “Verify and test” phase of pre-production is crucial for policy enforcement. Automating vulnerability scanning and security testing can not only save time, but also make the program better. Having a good automated system can reduce the risk of error on the tester’s side. If the automation is set up correctly it can have the standards in mind and test the program accordingly. Having a good, automated tester is crucial for making sure the project meets the standards.

Another place where automation should occur is in the “Monitor and detect” phase of production. Log collection and event alerting are both things that can be easily automated. Notifications and logs can be automatically created if the program does not meet the standards. Also the program should automatically be able to prevent/stop any chance of the program going against the guidelines.

### 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-C | Low | Unlikely | High | P1 | L3 |
| STD-002-C | High | Probable | High | P6 | L2 |
| STD-003-C | Medium | Unlikely | Medium | P4 | L3 |
| STD-004-J | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-006-C | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | Low | Likely | Medium | P6 | L2 |
| STD-009-CPP | Low | Unlikely | Medium | P2 | L3 |
| STD-010-J | Low | Unlikely | Medium | P2 | L3 |

### 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 at rest | Encryption at rest refers to encrypting data that remains unused. For example, if you have a database containing usernames, passwords, or other potentially sensitive information not in continuous use, it's very important to encrypt and safeguard that data. |
| Encryption in flight | Encryption in flight refers to the encryption of data that is coming in or out of the system. For example, sending emails is considered encryption in transit. It means that data needs to be encrypted on its whole journey from system to system. |
| Encryption in use | Encryption in use refers to the encryption of data that is currently being used by the user. This makes it so there is encryption where data is created, viewed, and edited. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to the process of verify the user who is trying to access the system, or certain parts of the system. There should be a username and password for each user that can be verified as to what their privileges are. |
| Authorization | Authorization is the process of making sure and giving the right privileges to the user after they have been authenticated. An administrative user should be able to access the database so they can update it if necessary. This includes adding users if necessary. |
| Accounting | Accounting refers to the process of logging important information for the benefit of the system. For example, a log file that records the activity of the system would be accounting. This is a big part of security as these logs can give valuable information about if the system was accessed by an unauthorized part and can shed light on how to fix this. |

**\***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.

STD-001-C: 1. “Validate Input data”: We want to make sure that we are implementing abstract data types using opaque data types.

STD-002-C: 1. “Validate Input Data”: We want to make sure that after an int conversion that all the data is intact.

STD-003-C: 10. “Adopt a Secure Coding Standard”: We want to make sure that we catch errors. We need proper standards to set up measures to do this.

STD-004-J: 3. “Architect and Design for Security Policies”: This applies because we are making sure that an SQL injection does not occur.

7. “Sanitize Data Sent to Other Systems” this applies because we want to make sure that data sent to another system will not compromise the data or the system. This could cause an unexpected outcome.

STD-005-CPP: 2. “Heed Compiler Warnings” this applies because data races will cause a big mess by having two or more sources accessing the same memory location.

10. “Adopt a Secure Coding Standard” this applies because with a good coding standard this can be avoided.

STD-006-C: 10. “Adopt a Secure Coding Standard” this applies because using the correct assertion is about how well the programmer knows about them. With a good standard they will know what to do.

STD-007-CPP: 10. “Adopt a Secure Coding Standard” this applies because you have to make sure that you are following the standard. Otherwise, there will be problems with the program. A good standard will prevent this from happening.

STD-008-CPP: 9. “Use Effective Quality Assurance Techniques” this applies because using good testing during QA will make sure this is caught and fixed.

STD-009-CPP: 9. “Use Effective Quality Assurance Techniques” this applies because during QA this should be caught and Fixed.

10. “Adopt a Secure Coding Standard” this applies because with a good standard this should not occur.

STD-010-J: 10. “Adopt a Secure Coding Standard” this applies because with a good standard there should not be an empty infinite loop.

**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 | 03/19/2024 | Milestone completion | Liam Sullivan | Kaan Esendemir |
| 1.2 | 04/12/2024 | Project Completion | Liam Sullivan | Kaan Esendemir |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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