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

Michael Neff



# 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/" \o "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 | Validating input is a way to check that the data received is free of malicious or harmful intents. SQL injections can be embedded in received data and by validating these possible risks, it prevents and reduces the potential security risks on the data being inputted |
| 1. Heed Compiler Warnings | Heed compiler warnings are alerts presented when there are potential code issues. These warnings can show potential bugs and vulnerabilities. These are considered best coding practices and ignoring these warnings can lead to unsecure code. Allowing possible threats to be executed. |
| 1. Architect and Design for Security Policies | These policies involve creating systems and software where security guidelines and best practices are used. Ensuring that security systems are implemented at the start of the project rather than having to implement the security measures halfway or at the end of the project helps reduce risk and compiles with the standards from the start. This ensures outdated phrases and codes are corrected enhancing the protection of possible injections. |
| 1. Keep It Simple | Keeping it simple is simple. Ensuring simple implementation and design tasks and also reducing complexity helps audits in the long run. Maintenance-free is key, and making sure we have straightforward and maintainable code avoids future confusion of features that can introduce any find of risk. |
| 1. Default Deny | Default deny lowers the risk of unauthorized access by ensuring the authentication process grants rather than assuming a user has access until told else. |
| 1. Adhere to the Principle of Least Privilege | Principles of least privilege is a guide for minimum-level access for systems applications and even users. The limitations of access reduce the potential impact of a security breach and adversely, minimize risks or misuse. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing is a step that uses multiple layers of security to protect data. Implementing this security method at different endpoints is an approach that ensures each layer is passing through safe data and that no one layer is compromised. This is a great strategy for protecting and overall enhancing program security. |
| 1. Practice Defense in Depth | This principle also uses multiple layers of security controls to protect data. Implementing the measures at all levels like network, endpoint, and even application provides protection and overall program security. |
| 1. Use Effective Quality Assurance Techniques | QA is a great technique and a big part of all programs. The testing and evaluation of code that ensures the software meets the requirements and security standards. Code reviews and automated testing along with vulnerability assessments, and out-of-the-box thinking allows a QA to ensure the code is safe, concise, and free of bugs. |
| 1. Adopt a Secure Coding Standard | Security coding standards are standards that follow the best practices for writing code and minimizing any security risks. This principle helps delvers avoid common vulnerabilities and maintain consistency in code security on all projects. |

### 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** | DCL12-C. | Implement abstract data types using opaque types |

**Cited:**

**<https://wiki.sei.cmu.edu/confluence/display/c/DCL12-C.+Implement+abstract+data+types+using+opaque+types>**

| **Noncompliant Code** |
| --- |
| This noncompliant code example is based on the managed string library developed by CERT [Burch 2006]. In this example, the managed string type and the functions that operate on this type are defined in the string\_m.h header file as follows: |
| 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** |
| --- |
| In the external string\_m.h file, the string\_mx type is defined to be an instance of struct string\_mx, which in turn is declared as an incomplete 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | UNLIKELY | HIGH | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | 7.2.0 | CertC-DCL12 | This Tool checks for Implementation where pointers to this object are not dereference. |
| LDRA tool suite | 9.7.1 | 104 D | Partially implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | MSC33-C | Do not pass invalid data to the asctime() function |

**Cited:**

**<https://wiki.sei.cmu.edu/confluence/display/c/MSC33-C.+Do+not+pass+invalid+data+to+the+asctime%28%29+function>**

| **Noncompliant Code** |
| --- |
| This noncompliant code example invokes the asctime() function with potentially unsanitized data |
| #include <time.h>    void func(struct tm \*time\_tm) {  char \*time = asctime(time\_tm);  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The strftime() function allows the programmer to specify a more rigorous format and also to specify the maximum size of the resulting time string: |
| #include <time.h>    enum { maxsize = 26 };    void func(struct tm \*time) {  char s[maxsize];  /\* Current time representation for locale \*/  const char \*format = "%c";    size\_t size = strftime(s, maxsize, format, time);  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | Likely | LOW | P27 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | BADFUNC.TIME\_H | Use of <time.h> Time/Date Function |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced Enforcement |
| PC-lint Plus | 1.4 | 586 | Fully Supported |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | ERR34-C | Detect errors when converting a string to a number |

**Cited:**

**<https://wiki.sei.cmu.edu/confluence/display/c/ERR34-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: |
| 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | cert-err34-c | Checked by clang-tidy |
| Polyspace Bug Finder | R2024a | CERT C: Rule ERR34-C | Checks for unsafe conversion from string to numeric value |
| SonarQube C/C++ Plugin | 3.11 | S989 |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | IDS00-J. | Prevent SQL injection |

**Cited:**

**<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88487713>**

| **Noncompliant Code** |
| --- |
| The password argument cannot be used to attack this program because it is passed to the hashPassword() function, which also sanitizes the input. |
| 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | S2007 | Executing SQL queries is security-sensitive |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| CodeSonar | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | CON52-CPP | Prevent data races when accessing bit-fields from multiple threads |

**Work Cited:**

**<https://wiki.sei.cmu.edu/confluence/display/cplusplus/CON52-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. |
| #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;  } |

| **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 |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | read\_write\_data\_race | Supported |
| CodeSonar | 8.1p0 | CONCURRENCY.DATARACE | Data Race |
| Coverity] | 6.5 | RACE\_CONDITION | Fully implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-CON52-a | Use locks to prevent race conditions when modifying bit fields |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | DCL03-C | Use a static assertion to test the value of a constant expression |

**Work Cited:**

**<https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory>**

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly: |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution: |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | clang-analyzer- | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 8.1p0 | ALLOC.UAF | Use after free |
| LDRA tool suite | 9.7.1 | 483 S, 484 S | Partially implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR60-CPP | Exception objects must be nothrow copy constructible |

**Work Cited:**

**<https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR60-CPP.+Exception+objects+must+be+nothrow+copy+constructible>**

| **Noncompliant Code** |
| --- |
| 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** |
| --- |
| 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**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 | 2024.2 | C++3508 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR60-a  CERT\_CPP-ERR60-b | Exception objects must be nothrow copy constructible |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | OOP52-CPP | Do not delete a polymorphic object without a virtual destructor |

**Work Cited:**

**<https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP52-CPP.+Do+not+delete+a+polymorphic+object+without+a+virtual+destructor>**

| **Noncompliant Code** |
| --- |
| The implicitly declared destructor is not declared as virtual even in the presence of other virtual functions. |
| struct Base {  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the destructor for Base has an explicitly declared virtual destructor, ensuring that the polymorphic delete operation results in well-defined behavior. |
| struct Base {  virtual ~Base() = default;  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | non-virtual-public-destructor-in-non-final-class | Partially Checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.DNVD | Delete with Non-Virtual Destructor |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-OOP52-a | Define a virtual destructor in classes used as base classes which have virtual functions |
| RuleChecker | 22.10 | non-virtual-public-destructor-in-non-final-class | Partially Checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | CON50-CPP | Do not destroy a mutex while it is locked |

**Work Cited:**

**<https://wiki.sei.cmu.edu/confluence/display/cplusplus/CON50-CPP.+Do+not+destroy+a+mutex+while+it+is+locked>**

| **Noncompliant Code** |
| --- |
| Unfortunately, this code contains a race condition, allowing the mutex to be destroyed while it is still owned, because start\_threads() may invoke the mutex's destructor before all of the threads have exited. |
| #include <mutex>  #include <thread>    const size\_t maxThreads = 10;    void do\_work(size\_t i, std::mutex \*pm) {  std::lock\_guard<std::mutex> lk(\*pm);    // Access data protected by the lock.  }    void start\_threads() {  std::thread threads[maxThreads];  std::mutex m;    for (size\_t i = 0; i < maxThreads; ++i) {  threads[i] = std::thread(do\_work, i, &m);  }  } |

| **Compliant Code** |
| --- |
| This compliant solution eliminates the race condition by extending the lifetime of the mutex. |
| #include <mutex>  #include <thread>    const size\_t maxThreads = 10;    void do\_work(size\_t i, std::mutex \*pm) {  std::lock\_guard<std::mutex> lk(\*pm);    // Access data protected by the lock.  }    std::mutex m;    void start\_threads() {  std::thread threads[maxThreads];    for (size\_t i = 0; i < maxThreads; ++i) {  threads[i] = std::thread(do\_work, i, &m);  }  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p.0 | CONCURRENCY.LOCALARG | Local Variable Passed to Thread |
| Helix QAC | 2024.1 | DF961, DF4962 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-CON50-a | Do not destroy another thread's mutex |
| Polyspace Bug Finder | R2024a | CERT C++: CON50-CPP | Checks for destruction of locked mutex (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | FIO51-CPP | Close files when they are no longer needed |

**Work Cited:**

**<https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed>**

| **Noncompliant Code** |
| --- |
| 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2024.2 | DF4786, DF4787, DF4788 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Polyspace Bug Finder | R2024a | CERT C++: FIO51-CP | Checks for resource 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.

By using the toolchain we can view the access and plan phases. This is where DevSecOps comes into play. The design and build phases are where security is addressed. Testing to ensure the automated scans for verification and test are integrated and other testing that may need to be done, especially compliance testing. Once the tests continue network monitoring and testing of performance logs and more are done, this is just to define simple threat detection’s. All of these should be done early on in the building process.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| **Rule** | **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- | --- |
| **STD-001-CPP** | High | Unlikely | Medium | High | 2 |
| DCL12-C. | Low | Unlikely | High | P1 | L3 |
| MSC33-C | Low | Likely | Low | P27 |  |
| ERR34-C | Low | Unlikely | Medium | P4 | L3 |
| IDS00-J | Low | Likely | Medium | P18 | L1 |
| CON52-CPP | High | Probable | Medium | P8 | L2 |
| DCL03-C | High | Likely | Medium | P18 | L1 |
| ERR60-CPP | High | Likely | Medium | p18 | L1 |
| ERR60-CPP | Low | Probable | Medium | P4 | L3 |
| OOP52-CPP | Low | Likely | Low | P9 | L2 |
| CON50-CPP | Medium | Probable | High | P4 | L3 |
| FIO51-CPP | Medium | Unlikely | Medium | P4 | 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 rest is data that is stored on a physical memory device, like hard drives, SSD’s and even as far as cloud storage. The encryption at rest protects data from an unauthorized breach. It plays a crucial role for defending sensitive information from exposure, especially when a device becomes lost. |
| Encryption in flight | Data that is being transferred from one location over a network used encryption in flight. The end game is that encryption in flight is protecting data against interception while being transmitted. |
| Encryption in use | This process is unlike encryption at rest and flight. This encryption protects data specific to states, and is used to secure data while in use. This protects against threats such as unauthorized or non authenticated breaches, and or memory scraping. This form of encryption ensures that information remains protected. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is verification of identify of users or systems before access is granted. There are many ways to authenticate a user. Unique passwords, biometric and pin codes, multi factor authentication is also common. These steps ensure that the information is correct and can provide access to anyone who is authenticate. |
| Authorization | Authorization is the process after the authentication verifies identity. Authorization defines the levels of permissions granted to a user. This can be done to specific roles if relating to a work place system. |
| Accounting | Accounting involves tracking and recording of users data. This includes logging, file access, and changes to usage. This is essential for monitoring security compliance and it also helps ensure that policies and security’s are in place. |

**\***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** | 07/25/2024 | Moudle 3 | Michael Neff |  |
| **1.2** | 08/11/2024 | Moudle 6 Project | Michael Neff |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

|  |
| --- |
| 1. ValidateInput Data |
| 1. Heed Compiler Warnings |
| 1. Architect and Design for Security Policies |
| 1. Keep It Simple |
| 1. Default Deny |
| 1. Adhere to the Principle of Least Privilege |
| 1. Sanitize Data Sent to Other Systems |
| 1. Practice Defense in Depth |
| 1. Use Effective Quality Assurance Techniques |
| 1. Adopt a Secure Coding Standard |

ValidateInput Data,Heed Compiler Warnings, Architect and Design for Security Policies, Keep It Simple, Default Deny, Adhere to the Principle of Least Privilege, Sanitize Data Sent to Other Systems, Practice Defense in, Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard