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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Input that has not been properly validated is a common cause of vulnerabilities. To properly validate input, you must identify input sources and specify what type of input the program is to receive. All input needs to be validated against the specifications put in place. |
| 1. Heed Compiler Warnings | It is best to ensure that the compiler being used has type-checking support on a higher level to detect mistakes automatically. Detecting more compiler warnings allows for the programmer to clean up the code along with providing a chance for the programmer to fix any minor problems that could leave the program vulnerable. This helps to prevent or lessen the chance of common exploits. |
| 1. Architect and Design for Security Policies | The architecture and design of a program plays a big role in how secure it is. The architecture should enforce security principles and can be done with the use of privileges and subsystems if those avenues are also protected. Secure design patterns can be utilized to help eliminate vulnerabilities that have been accidentally inserted into the code along with mitigating the consequences of said vulnerabilities. |
| 1. Keep It Simple | Complex designs, functions, and variables of a program can lead to there being vulnerabilities. Keeping them simple and straightforward will help to increase security. |
| 1. Default Deny | Access should be based on exclusions rather than permissions since it has the possibility to grant access to someone or something it shouldn’t. This issue can go unnoticed during normal use of a system. |
| 1. Adhere to the Principle of Least Privilege | A program and its users should operate with the lowest privileges possible. This means that processes should be executed with the minimum permission that is required to perform secure operations. If higher level permissions are needed for specific tasks, then permission can be granted and removed as soon as they’re no longer needed. |
| 1. Sanitize Data Sent to Other Systems | Sanitization is the process of ensuring that data sent to the subsystem conforms to the requirements whether that be for the subsystem or security related requirements. This prevents the leaking or exposure of sensitive data. Sanitization and input validation can be utilized together. |
| 1. Practice Defense in Depth | Defense in depth is the practice of utilizing multiple layers of security. If one layer proves to be inadequate then another layer will provide security to prevent or lessen the consequences of exploited vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Having multiple defect removal techniques throughout the development lifecycle can help to reduce the percentage of defects introduced during the lifecycle. These quality assurance techniques can help to reduce or even eliminate vulnerabilities that show up after release. |
| 1. Threat Modeling | Anticipating threats is part of creating secure software. Threat modeling involves determining what assets the system needs to protect, threats to those assets and their ranking, creating a security profile for the program, and creating a mitigation plan that’s implemented in the design, code and test cases for the software. |

### 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-CPP] | Abstract data types should be implemented using opaque or private data types. |

| **Noncompliant Code** |
| --- |
| Implementation of string\_mx is visible to user meaning they may end up directly manipulating fields. Example from https://wiki.sei.cmu.edu/confluence/display/c/DCL12-C.+Implement+abstract+data+types+using+opaque+types |
| **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** |
| --- |
| String mx is changed to private to hide implementation of data type from user to prevent direct manipulation. Example from https://wiki.sei.cmu.edu/confluence/display/c/DCL12-C.+Implement+abstract+data+types+using+opaque+types |
| **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):** Principle 4 – Proper utilization of private variables/structures will keep the main section of code cleaner, easier to follow, and more simple. |
| --- |

**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-DCL12 |  |
| LDRA Tool Suite | 9.7.1 | 104 D | Partially Implemented |
| Polyspace Bug Finder | R2024a | CERT C: Rec. DCL12-C | Checks for structure and object implementation |
| Parasoft C/C++ test | 2023.1 | CERT\_C-DCL 12-a | Pointers to structures that are not dereferenced within translation units then the object should have hidden implementation |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CLG] | Conversions of data must guarantee that there is no lost or misinterpreted data. |

| **Noncompliant Code** |
| --- |
| Loss of data and sign can occur when converting from an unsigned int to signed int. |
| #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** |
| --- |
| Proper implementation of conversion of unsigned to signed int |
| #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):** Principle 1 – Data cannot be verified properly if it isn’t implemented correctly |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  |  |
| CodeSonar | 8.1p0 | LANG.CAST.PC.AV LANG.CAST.PC.CONST2PTR LANG.CAST.PC.INT  **LANG.CAST.COERCE LANG.CAST.VALUE**  **ALLOC.SIZE.TRUNC MISC.MEM.SIZE.TRUNC**  **LANG.MEM.TBA** |  |
| Coverity | 2017.07 | NEGATIVE\_RETURNS  REVERSE\_NEGATIVE  MIRSE\_CAST | Can look for expressions that may contain dangerous implied int conversions |
| CPPcheck | 1.66 | memsetValueOutOfRange | Memset() cannot be represented as an unsigned char |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP | Std::string should not be created from a null pointer |

| **Noncompliant Code** |
| --- |
| Std::string object is created from a call to std::getenv() which returns a null pointer on failure and can cause undefined behavior |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| Call to std::getenv() is checked to see if it’s null before std::string object is created |
| #include <cstdlib>  #include <string>    void f() {  const char \*tmpPtrVal = std::getenv("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** Validate input data – Prevent unexpected behavior from code by verifying data before applying to string |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Assert\_failer |  |
| CodeSonar | 8.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Klocwork | 2024.1 | DF4770, DF4771, DF4772, DF4773, DF4774 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP\_STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAV | SQL injection vulnerabilities appear in applications that take elements of a SQL query from an untrusted source |

| **Noncompliant Code** |
| --- |
| Allows for SQL injection attack by incorporating an unsanitized input argument “username” into the prepared statement |
| **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;        PreparedStatement stmt = connection.prepareStatement(sqlString);          ResultSet rs = stmt.executeQuery();  **if** (!rs.next()) {  **throw** **new** SecurityException("User name or password incorrect");        }          // Authenticated; proceed      } **finally** {  **try** {          connection.close();        } **catch** (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| Uses a parametric query with ? character as a placeholder. Validates length of “username” argument |
| **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):** Architecture and design – When utilizing SQL queries you need to verify the sources used |
| --- |

**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 |
| 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** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-JAV] | Limit the lifetime of sensitive data |

| **Noncompliant Code** |
| --- |
| Code reads username and password from the console and stores password in a string. Remains exposed until garbage collector reclaims memory |
| **class** Password {  **public** **static** **void** main (String args[]) **throws** IOException {      Console c = System.console();  **if** (c == **null**) {        System.err.println("No console.");        System.exit(1);      }        String username = c.readLine("Enter your user name: ");      String password = c.readLine("Enter your password: ");    **if** (!verify(username, password)) {  **throw** **new** SecurityException("Invalid Credentials");      }        // User is authorized, continue...    }      // Dummy verify method, always returns true  **private** **static** **final** **boolean** verify(String username, String password) {  **return** **true**;    }  } |

| **Compliant Code** |
| --- |
| Uses Console.readPassword() to obtain password from console |
| **class** Password {  **public** **static** **void** main (String args[]) **throws** IOException {      Console c = System.console();    **if** (c == **null**) {        System.err.println("No console.");        System.exit(1);      }        String username = c.readLine("Enter your user name: ");  **char**[] password = c.readPassword("Enter your password: ");  **boolean** isValidUser = verify(username, password);        // Clear the password      Arrays.fill(password,' ');    **if** (!isValidUser) {  **throw** **new** SecurityException("Invalid Credentials");      }        // User is authorized, continue...    }      // Dummy verify method, always returns true  **private** **static** **final** **boolean** verify(String username, **char**[] password) {  **return** **true**;    }  } |

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

| **Principles(s):** Architecture and design – Protection of data in memory requires support from the underlying operating system |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Jtest | 2023.1 | CERT.MSC04.LEAKS | Ensure resources are deallocated |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| Uses assert() macro to assert a property concerning a memory-mapped structure which 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** |
| --- |
| Uses static\_assert() which allows for incorrect assumptions to be diagnosed at compile instead of resulting in a silent malfunction or runtime error |
| #include <assert.h>    **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    static\_assert(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**),                "Structure must not have any padding"); |

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

| **Principles(s): Use effective quality assurance techniques – Testing sections of code instead of massive blocks allows for the early detection of bugs and errors** |
| --- |

**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 |
| Éclair | 1.2 | CC2.DCL03 | Fully Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| Neither f() nor main() catch exceptions thrown by throwing\_func(). Since no matching handler is found, std::terminate() is called |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| Main entry point handles exceptions |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):** Use effective quality assurance techniques – Abnormal termination is a typical vectore for denial-of-service attacks. Exceptions thrown by an application must be caught by a matching handler even if the exception cannot be gracefully recovered from. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all-early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 |  |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable catch |
| LADRA tool suite | 9.7.1 | 527 S | Partially implemented |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CLG] | Floating point numbers should be avoided in situations where precise computation is needed |

| **Noncompliant Code** |
| --- |
| Takes the mean of 10 identical numbers and checks to see if it matches but because of the imprecision of floating point arithmetic the computed mean doesn’t match |
| #include <stdio.h>    /\* Returns the mean value of the array \*/  **float** mean(**float** array[], **int** size) {  **float** total = 0.0;  **size\_t** i;  **for** (i = 0; i < size; i++) {      total += array[i];  **printf**("array[%zu] = %f and total is %f\n", i, array[i], total);    }  **if** (size != 0)  **return** total / size;  **else**  **return** 0.0;  }    **enum** { array\_size = 10 };  **float** array\_value = 10.1;    **int** main(**void**) {  **float** array[array\_size];  **float** avg;  **size\_t** i;  **for** (i = 0; i < array\_size; i++) {      array[i] = array\_value;    }      avg = mean( array, array\_size);  **printf**("mean is %f\n", avg);  **if** (avg == array[0]) {  **printf**("array[0] is the mean\n");    } **else** {  **printf**("array[0] is not the mean\n");    }  **return** 0;  } |

| **Compliant Code** |
| --- |
| Replace the floating point numbers with integers for internal additions |
| #include <stdio.h>    /\* Returns the mean value of the array \*/  **float** mean(**int** array[], **int** size) {  **int** total = 0;  **size\_t** i;  **for** (i = 0; i < size; i++) {      total += array[i];  **printf**("array[%zu] = %f and total is %f\n", i, array[i] / 100.0, total / 100.0);    }  **if** (size != 0)  **return** ((**float**) total) / size;  **else**  **return** 0.0;  }    **enum** {array\_size = 10};  **int** array\_value = 1010;    **int** main(**void**) {  **int** array[array\_size];  **float** avg;  **size\_t** i;  **for** (i = 0; i < array\_size; i++) {      array[i] = array\_value;    }      avg = mean(array, array\_size);  **printf**("mean is %f\n", avg / 100.0);  **if** (avg == array[0]) {  **printf**("array[0] is the mean\n");    } **else** {  **printf**("array[0] is not the mean\n");    }  **return** 0;  } |

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

| **Principles(s):** Validate input data – Improper validation of data can lead to errors within the code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FLP02 | Partially checked |
| Helix QAC | 2024.1 | C0790 |  |
| LDRA tool suite | 9.7.1 | 56 S | Partially implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_C\_FLP02-a | Floating point expressions shall not be tested for equality or inequality |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CLG] | Signal handler persistence requires understanding of implementation-specific details |

| **Noncompliant Code** |
| --- |
| Fails to persist signal handler on Windows and Unix platforms |
| **void** handler(**int** signum) {    /\* Handle signal \*/  } |

| **Compliant Code** |
| --- |
| POSIX sigation() assigns handlers to signals and also allows signal-handler persistence to be controlled via SA\_RESETHAND flag |
| /\*   \* Equivalent to signal(SIGUSR1, handler) but makes   \* signal persistent.   \*/  **struct** sigaction act;  act.sa\_handler = handler;  act.sa\_flags = 0;  **if** (sigemptyset(&act.sa\_mask) != 0) {    /\* Handle error \*/  }  **if** (sigaction(SIGUSR1, &act, NULL) != 0) {    /\* Handle error \*/  } |

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

| **Principles(s):** Default deny – Denying improperly handled signals will provide more security for the program |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | BADFUNC.SIGNAL | Use of signal |
| Helix QAC | 2024.1 | C5020 |  |
| LDRA tool suite | 9.7.1 | 97 D | Partially implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_C-SIG01-a | Singal handling facilities of <signal.h> shall not be used |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CGL] | Functions should have valid parameters |

| **Noncompliant Code** |
| --- |
| Setfile() and usefile() do not validate parameters. Invalid file pointers can expose a vulnerability |
| /\* Sets some internal state in the library \*/  **extern** **int** setfile(**FILE** \*file);    /\* Performs some action using the file passed earlier \*/  **extern** **int** usefile();    **static** **FILE** \*myFile;    **void** setfile(**FILE** \*file) {      myFile = file;  }    **void** usefile(**void**) {      /\* Perform some action here \*/  } |

| **Compliant Code** |
| --- |
| Validation of function parameters and verification of the internal state leads to consistency in program execution and can eliminate potential vulnerabilities |
| /\* Sets some internal state in the library \*/  **extern** errno\_t setfile(**FILE** \*file);    /\* Performs some action using the file passed earlier \*/  **extern** errno\_t usefile(**void**);    **static** **FILE** \*myFile;    errno\_t setfile(**FILE** \*file) {  **if** (file && !**ferror**(file) && !**feof**(file)) {      myFile = file;  **return** 0;    }      /\* Error safety: leave myFile unchanged \*/  **return** -1;  }    errno\_t usefile(**void**) {  **if** (!myFile) **return** -1;        /\*       \* Perform other checks if needed; return       \* error condition.       \*/        /\* Perform some action here \*/  **return** 0;  } |

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

| **Principles(s):** Validate Input Data – Data should always be verified along with anything that is handling said data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.STRUCT.UPD | Unchecked parameter deference |
| Parasoft C/C++ test | 2023.1 | CERT\_C-API00-a | The validity of parameters much be checked inside each function |
| PC-lint Plus | 1.4 | 413, 613, 668 | Partially supported, reports null pointers including function parameters |
| PVS-Studio | 7.31 | V781 |  |

### 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.

Automated testing is a process that can be included in the process at the beginning. The Planning stage should include creating a plan for automation and where they can use it. During the design phase they should be designing tests based on assessed threats and errors. The actually testing will then occur during the building phase so that code can be tested as soon as possible to fix any common or small errors that could show up later down the line. Including automation throughout the entirety of the provided process will help to ensure the overall security of the program.

### 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 | Low | Unlikely | High | Low | 2 |
| STD-002-CLG | High | Probably | High | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-JAV | High | Likely | Medium | High | 1 |
| STD-005-JAV | High | Probable | Medium | High | 1 |
| STD-006-CGL | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Probable | High | Low | 3 |
| STD-008-CGL | Low | Probable | High | Low | 3 |
| STD-009-CGL | Low | Unlikely | Low | Low | 3 |
| STD-010-CGL | Medium | Unlikely | High | Low | 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 at rest | Encryption at rest refers to data that is encrypted while stored to prevent unauthorized access. In order to access the data, the individual or system will need the key. Encryption at rest is great for companies or organizations that utilize cloud services. |
| Encryption in flight | This refers to protecting data when it is being sent across a network by having it encrypted. This is especially important when sending data over open internet. Only the individual or system that is meant to receive the data should have the key. |
| Encryption in use | Encryption in use refers to when data is encrypted whenever it needs to be used. For example, when it is being input, processed, accessed, read, and updated. This allows for computations to run on encrypted files since decryption isn’t needed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to requiring any users or systems to verify they have permission to access another system, program, or even files. This can involve biometric scanners, usernames and passwords, and two-factor authentication. |
| Authorization | Authorization can be utilized with authentication to verify that a certain individual or system has access to specific parts of the system. The authorization level will depend on the level of security in place. Individuals or systems may need authorization for something like specific files or it can mean they need to gain prior authorization to make changes dealing with the database. |
| Accounting | Accounting refers to keeping track of when data is accessed, by who, for how long, what it was used for, etc. Accounting follows authentication and authorization within the timeline since the individual will need to authenticate that they have access and have the appropriate level of authorization to access the data. |

**\***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 |  |
| 1.5 | 5/25/2024 | Standards and Principles | Keri Ludemann | [Insert text.] |
| 2.0 | 6/15/2024 | Complete | Keri Ludemann | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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