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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_gjdgxs)

[Purpose 2](#_30j0zll)

[Scope 2](#_1fob9te)

[Module Three Milestone 2](#_3znysh7)

[Ten Core Security Principles 2](#_2et92p0)

[C/C++ Ten Coding Standards 3](#_3dy6vkm)

[Coding Standard 1 4](#_1t3h5sf)

[Coding Standard 2 5](#_4d34og8)

[Coding Standard 3 6](#_2s8eyo1)

[Coding Standard 4 7](#_17dp8vu)

[Coding Standard 5 8](#_3rdcrjn)

[Coding Standard 6 9](#_26in1rg)

[Coding Standard 7 10](#_lnxbz9)

[Coding Standard 8 11](#_35nkun2)

[Coding Standard 9 13](#_1ksv4uv)

[Coding Standard 10 14](#_44sinio)

[Defense-in-Depth Illustration 15](#_2jxsxqh)

[Project One 15](#_z337ya)

[1.](#_3j2qqm3) Revise the C/C++ Standards 15

[2.](#_1y810tw) Risk Assessment 15

[3.](#_4i7ojhp) Automated Detection 15

[4.](#_2xcytpi) Automation 15

[5.](#_1ci93xb) Summary of Risk Assessments 16

[6.](#_3whwml4) Create Policies for Encryption and Triple A 16

[7.](#_2bn6wsx) Map the Principles 17

[Audit Controls and Management 18](#_qsh70q)

[Enforcement 18](#_3as4poj)

[Exceptions Process 18](#_1pxezwc)

[Distribution 19](#_49x2ik5)

[Policy Change Control 19](#_2p2csry)

[Policy Version History 19](#_147n2zr)

[Appendix A Lookups 19](#_3o7alnk)

[Approved C/C++ Language Acronyms 19](#_23ckvvd)

## 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 | This principle describes the idea that all user input should be validated to prevent unauthorized access to systems and data. This prevents malicious breaches such as SQL injection and buffer overflows as well as unexpected and unintended responses. |
| 1. Heed Compiler Warnings | Compilers provide useful information such as access logs, warnings, and error codes. It is impossible for a developer to anticipate all of the possible scenarios that might arise upon running code, but the compiler will provide information as it is compiling the code. This information could be known vulnerabilities and suggestions on how to fix code that is insecure. |
| 1. Architect and Design for Security Policies | During the planning and design phase, keeping security policies in mind will aid the developer in ensuring vulnerabilities are accounted for and limited. Choosing an archetype that is security-centered will help keep user-data and systems secure in the case of intended attacks. |
| 1. Keep It Simple | It is more difficult to ensure that all aspects of complex systems are completely secure as there are many factors and variables that can be targeted and exploited. Keeping code as simple as possible will ensure that data and systems are easy-to-use and secure. |
| 1. Default Deny | This policy refers to the idea of denying access to a system to everyone by default unless specific rules and authentication measures are met. This will prevent unauthorized and malicious intentions from being carried out by would-be hackers. |
| 1. Adhere to the Principle of Least Privilege | Granting wide-spread access to even authorized users is unwise. These privileges are what is considered administrative access. Users should only be allowed to interact with parts of the system intended for their use as laid out by the archetype specifications which by the |
| 1. Sanitize Data Sent to Other Systems | Users can access data by overflowing data buffers and by passing data intended to execute queries into the code known as SQL injections. These types of attacks can be mitigated by sanitizing input data. Data sanitation describes the process of ensuring that data being passed is of the expected type and size as required by the function or system. This can involve ensuring queries are parameterized or making sure integer data is not large enough to overflow a buffer. |
| 1. Practice Defense in Depth | Defense in Depth(DiD) refers to the concept of protecting data behind multiple layers of security. These layers can come in the form of software defense such as firewalls and proxy servers, but also physical protections such as locked server rooms and cabinets. Bundling security layers offers higher levels of security than any single security layer alone. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance describes ensuring that code is up to outlined standards and principles. These actions could entail checking and testing code to address security vulnerabilities and confirming that everything works exactly as intended by the developers and the software design document. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard will aid the designers and developers in creating secure architectures and code that will be the most efficient at preventing breaches while offering the highest capability and most appropriate functionality as outlined by the project specifications. |

### 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** | **Implement Abstract Data Types Using Opaque Types** |
| --- | --- | --- |
| **Data Type** | [DCL-012-C] | Opaque types hide the implementation functions from the user in a bid to prevent unauthorized manipulation of the data type. This method only provides enough information to the user to implement, refer to, and manipulate the values of the data type without seeing the specific functions that define the implementation, which avoids unauthorized manipulation of the data type’s implementation. |

| **Noncompliant Code** |
| --- |
| The below example provides a data type’s implementation structure to the user, which could allow the user to directly manipulate the data type using its inherent functions and structure. |
| **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** |
| --- |
| The below code hides the data type’s structure and implementation functions from the user, which minimizes the possibility of direct manipulation. |
| **External Header File:**  **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);  /\* ... \*/  **Internal Header File:**  **struct** string\_mx {  **size\_t** size;  **size\_t** maxsize;  unsigned **char** strtype;  **char** \*cstr;  }; |

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

| **Principles(s):** 8: Abstracting class information reduces the risk of exploitation through exposed methods and attributes. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion | 7.2.0 | CertC-DCL12 | [Insert text.] |
| LDRA tool suite | 9.7.1 | 104 D | Partially implemented |
| Polyspace Bug Finder | R2023a | CERT C: Rec. 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 | 2022.2 | 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** | **Ensure that Integer Conversions Don’t Result in Lost/Misrepresented Data** |
| --- | --- | --- |
| **Data Value** | [INT-031-C] | Converting data types to lower types can cause data to be truncated unexpectedly, such as when casting an unsigned integer value to a signed integer variable. |

| **Noncompliant Code** |
| --- |
| In the code below, the unsigned long integer value u\_a is assigned the maximum value for an unsigned integer value, which is 18,446,744,073,709,551,615. Then, this value is cast to a signed character data type, whose maximum value is 127, so the data is truncated considerably. |
| #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** |
| --- |
| In the code below, the unsigned integer value u\_a is assigned the maximum value for an unsigned integer value. Then it attempts to cast the data value to a signed character data type if the value is less than or equal to the maximum data value of that data type, otherwise an error is thrown. |
| #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):** 1: Validating input prevents buffer overflows through conversion issues; 3: Keeping security in mind during design will lead developers to pay attention to potential casting issues |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Klocwork | 2023.1 | PORTING.CAST.SIZE | [Insert text.] |
| TrustInSoft Analyzer | 1.38 | signed\_downcast | Exhaustively verified. |
| Polyspace Bug Finder | R2023a | CERT C: Rule INT31-C | Checks for:  Integer conversion overflow  Call to memset with unintended value  Sign change integer conversion overflow  Tainted sign change conversion  Unsigned integer conversion overflow  Rule partially covered. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Do Not Concatenate Different Types of String Literals** |
| --- | --- | --- |
| **String Correctness** | [STR-010-C] | Concatenating wide and narrow string literals leads to undefined behavior. |

| **Noncompliant Code** |
| --- |
| The code below attempts to concatenate a wide string literal with a narrow string literal. |
| **wchar\_t** \*msg = L"This message is very long, so I want to divide it "  "into two parts."; |

| **Compliant Code** |
| --- |
| The code below appropriately concatenates two wide string literals. |
| **wchar\_t** \*msg = L"This message is very long, so I want to divide it "  L"into two parts."; |

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

| **Principles(s):** 1: Validating input will prevent concatenation errors; |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | encoding-mismatch | Fully checked |
| Axivion | 7.2.0 | CertC-STR10 | [Insert text.] |
| ECLAIR | 1.2 | CC2.STR10 | Fully Implemented |
| RuleChecker | 22.04 | encoding-mismatch | Fully checked |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | [IDS-000-J] | Input data must always be validated in order to prevent malicious query manipulation and data leaks. |

| **Noncompliant Code** |
| --- |
| The code below allows the user to enter any string of any length directly into the query, potentially allowing buffer overflows and SQL query injections. |
| **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** |
| --- |
| The code below uses a parametric query using the character ‘?’ as a placeholder for user arguments. The code also prevents the user from entering a username and password that is unnecessarily long, which prevents overflow and extended queries that might contain injections. |
| **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):** 1: Validating input will reduce risk of injection attacks by preventing query injections through string input |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| CodeSonar | 7.3p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Findbugs | 1.0 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| SonarQube | 6.7 | S2077  S3649 | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Guarantee That Storage for Strings Has Sufficient Space for Character Data** |
| --- | --- | --- |
| **Memory Protection** | [STR-050-CPP] | Failing to ensure string input data has enough memory can have consequences such as buffer overflows, compromising the integrity of memory data. |

| **Noncompliant Code** |
| --- |
| The code below utilizes a bounded char array of size 12, which will truncate or overflow data in the case of given input passed that is above this limit. |
| #include <iostream>    **void** f() {  **char** buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The code below uses the standard library data type ‘std::string’ which automatically allocates the appropriate size for the input string. |
| #include <iostream>  #include <string>    **void** f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** 7: Sanitizing data to only allow strings of a certain size can prevent input being passed and lost 1: Validating input also ensures data is not too long for the variable to receive |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| Polyspace Bug Finder | R2023a | CERT C++: STR50-CPP | Checks for:  Use of dangerous standard function  Missing null in string array  Buffer overflow from incorrect string format specifier  Destination buffer overflow in string manipulation  Insufficient destination buffer size  Rule partially covered. |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Klocwork | 2023.1 | NNTS.MIGHT  NNTS.TAINTED  NNTS.MUST  SV.UNBOUND\_STRING\_INPUT.CIN | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a Static Assertion to Test the Value of a Constant Expression** |
| --- | --- | --- |
| **Assertions** | [DCL-003-C] | Assertions are great diagnostic tools to help locate vulnerabilities and errors in code at the time of compiling. |

| **Noncompliant Code** |
| --- |
| The code below incorrectly places an assertion within a function, which prevents the assertion from running unless the function is called, potentially missing valuable error-checking. |
| #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** |
| --- |
| The code below properly utilizes a static assertion statement to check data integrity at the time of compiling, alerting the developer of data overflow events when they occur. |
| #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):** 9: Unit testing ensures expected processing; 10: Testing code is a secure coding best practice |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | misc-static-assert | checked by clang-tidy |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | [Insert text.] |
| CodeSonar | 7.3p0 | (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** | **Prefer User-Defined Exceptions Over More General Exception Types** |
| --- | --- | --- |
| **Exceptions** | [ERR-051-J] | General exceptions can sometimes make errors difficult to diagnose. Specific, user-written exceptions can provide more information about where and why errors occur in the code. |

| **Noncompliant Code** |
| --- |
| The code below throws a general ‘cannot find file’ error upon compiling, despite ‘file not found’ defined as one of the switch cases. This occurs because the error received does not exactly match the switch case string. |
| **try** {  doSomething();  } **catch** (Throwable e) {  String msg = e.getMessage();  **switch** (msg) {  **case** "file not found":  // Handle error  **break**;  **case** "connection timeout":  // Handle error  **break**;  **case** "security violation":  // Handle error  **break**;  **default**: **throw** e;  }  } |

| **Compliant Code** |
| --- |
| The code below is hard-coded with specific try-catch statements that force the code to throw the appropriate exception upon running into an error. |
| **public** **class** TimeoutException **extends** Exception {  TimeoutException () {  **super**();  }  TimeoutException (String msg) {  **super**(msg);  }  }  // ...  **try** {  doSomething();  } **catch** (FileNotFoundException e) {  // Handle error  } **catch** (TimeoutException te) {  // Handle error  } **catch** (SecurityException se) {  // Handle error  } |

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

| **Principles(s):** 10: A common security standard is to use user-defined exception handlers so that exceptions are caught in situation specific to the code |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Unlisted | Unlisted | Unlisted | Unlisted | Unlisted |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Jtest | 2022.2 | CERT.ERR51.NCE | Do not catch exception types which are too general or are unchecked exceptions |
| SonarQube | 6.7 | S1193 | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do Not Depend on Undefined Behavior** |
| --- | --- | --- |
| **Testing** | [MSC-015-C] | Depending on undefined behavior can have unexpected results when running code, even when utilizing assertions. |

| **Noncompliant Code** |
| --- |
| The code below utilizes an assertion to test the value of a, but incorrectly assumes that ‘a’ will not overflow its buffer within the assertion, providing an unexpected result. |
| #include <assert.h>  #include <limits.h>  #include <stdio.h>    **int** foo(**int** a) {  **assert**(a + 100 > a);  **printf**("%d %d\n", a + 100, a);  **return** a;  }    **int** main(**void**) {  foo(100);  foo(INT\_MAX);  **return** 0;  } |

| **Compliant Code** |
| --- |
| The code below tests a against the maximum value it can hold with the computation performed on the known variable ‘INT\_MAX’, instead of the unknown variable ‘a’, preventing an unexpected result due to overflow. |
| #include <assert.h>  #include <limits.h>  #include <stdio.h>    **int** foo(**int** a) {  **assert**(a < (INT\_MAX - 100));  **printf**("%d %d\n", a + 100, a);  **return** a;  }    **int** main(**void**) {  foo(100);  foo(INT\_MAX);  **return** 0;  } |

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

| **Principles(s):** 9: Ensuring that data is what you want it to be goes a long way in preventing security vulnerabilities and is a great quality assurance technique |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | [Insert text.] | Supported: Astree reports undefined behavior |
| LDRA tool suite | 9.7.1 | 48 D, 63 D, 84 D, 113 D, 5 Q, 64 S, 65 S, 100 S, 109 S, 156 S, 296 S, 324 S, 335 S, 336 S, 339 S, 412 S, 427 S, 465 S, 482 S, 497 S, 545 S, 587 S, 608 S, 642 S, 62 X, 63 X | PArtially implemented |
| Parasoft C/C++test | 2022.2 | CERT\_C-MSC15-a | Evaluation of constant unsigned integer expressions should not lead to a wrap-around |
| PVS-Studio | 7.24 | V772 | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Use Conservative File naming Conventions** |
| --- | --- | --- |
| Naming | [IDS-050-J] | Using unconventional characters in variables and file names such as spaces, dashes, and namespace prefixes can cause problems in the code. |

| **Noncompliant Code** |
| --- |
| The code below uses a backslash, potentially causing assignment errors on systems that have different rules for mapping. |
| File f = **new** File("A\uD8AB");  OutputStream out = **new** FileOutputStream(f); |

| **Compliant Code** |
| --- |
| The code below uses a universally accepted naming convention of just alphanumeric characters, preventing unintended naming assignments. |
| File f = **new** File("name.ext");  OutputStream out = **new** FileOutputStream(f); |

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

| **Principles(s):** 10: Using appropriate naming conventions prevents vulnerabilities and file-handling errors in compiling |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Use Comments Consistently and in a Readable Fashion** |
| --- | --- | --- |
| Documentation | [MSC-004-C] | Comments aid readers of the code in navigating the code in order to troubleshoot errors, understand functionality, and integrate their own changes for their desired functionality. |

| **Noncompliant Code** |
| --- |
| The code below is messy and hard to read, inevitably causing misunderstandings in understanding, proper manipulation, and navigation. |
| // \*/ /\* Comment, not syntax error \*/    f = g/\*\*//h; /\* Equivalent to f = g / h; \*/    //\  i(); /\* Part of a two-line comment \*/    /\  / j(); /\* Part of a two-line comment \*/      /\*//\*/ l(); /\* Equivalent to l(); \*/    m = n//\*\*/o  + p; /\* Equivalent to m = n + p; \*/    a = b //\*divisor:\*/c  +d; /\*  \* Interpreted as a = b/c + d; in c90  \* compiler and a = b + d; in c99 compiler.  \*/ |

| **Compliant Code** |
| --- |
| The code below uses appropriate commenting conventions, allowing for easy reading and navigation. |
| /\* Nice simple comment \*/    **int** i; /\* Counter \*/ |

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

| **Principles(s):** 9: Using comments appropriately offers excellent documentation and readability and is a best practice |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | mmline-comment  sline-comment  sline-splicing  smline-comment | Partially checked |
| GCC | 4.3.5 | [Insert text.] | Can detect violations of this rule when the -Wcomment flag is used |
| Polyspace Bug Finder | R2023a | CERT C: Rec. MSC04-C | Checks for use of /\* and // within a comment (rule partially covered) |
| PC-lint Plus | 1.4 | 1, 427, 602, 689, 853,  9059, 9060, 9066, 9259 | Fully Supported |

### 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.

A great way to automate security policy compliance is by using a tool such as CloudChekr, a software tool that monitors code and systems for security vulnerabilities and compliance. Within the code, developers should use vulnerability checkers such as cppcheck, which will analyze code and notify the user if there are issues that could potentially lead to security vulnerabilities and unexpected processing. Using access managers such as in Amazon Web Service(AWS) can automate authorization access authorization across the system, denying and allowing where appropriate. Additionally, having a security response automation protocol would be beneficial, such as shutting down the server or limiting access when certain activities are flagged.

### 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 |
| --- | --- | --- | --- | --- | --- |
| DCL-012-C | Low | Unlikely | High | P1 | L3 |
| INT-013-C | High | Probable | High | P6 | L2 |
| STR-010-C | Low | Probable | Medium | P4 | L3 |
| IDS-000-J | High | Probable | Medium | P12 | L1 |
| STR-050-P | High | Likely | Medium | P18 | L1 |
| DCL-003-C | Low | Unlikely | High | P1 | L3 |
| ERR-051-J | N/a | N/a | N/a | N/a | N/a |
| MSC-015-C-J | High | Likely | Medium | P18 | L1 |
| IDS-050-J | Medium | Unlikely | Medium | P4 | L3 |
| MSC-004-C | 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 in rest | All sensitive data must be encrypted on the server storage database. This will ensure that unauthorized users will not be able to read the data. |
| Encryption at flight | All transmitted data must be encrypted using standard transfer protocols such as SSL/TLS or SSH. This will secure the data and keep it unreadable by unintended parties in the event of interception. |
| Encryption in use | All data processed by the system must be secure to safeguard company, third-party, and client data. This will ensure no information is garnered by malicious parties. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | All personnel must have a secure password and not share the password with anyone. This ensures only authorized individuals will have access to the appropriate data. |
| Authorization | The system will utilize the principle of least privilege. This will ensure that users will only have access to data that is required for their duties and requirements. |
| Accounting | All activity on the server must be tracked and logged. This will help with monitoring for security vulnerabilities and suspicious activity. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 04/06/2023 | Completed Policy Guide | William West | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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