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

SARAM NADEEM



# 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 | This makes sure that all data which is received by a program or software becomes properly validated and sanitized so that it can’t be under threat of a security vulnerability. Proper implementation of data input validation can help in preventing buffer overflow and injection attacks. |
| 1. Heed Compiler Warnings | Paying attention to the compiler warnings during the development phase is vital. It can assist developers in discovering issues within their program or code which they might have missed. Catching these issues early on allows for a larger period of time to implement deeper and more secure security features within their programs. |
| 1. Architect and Design for Security Policies | Incorporating security policies for the architect and design of software development can greatly assist in implementing security procedures at every set of the actual development process. When following through with the architecture, having the policies set in place allows for a quicker and more efficient development process. |
| 1. Keep It Simple | Simplicity within code is a vital aspect of creating a system that can function properly and be further improved in the future. Complexity can create confusion between developers who might be working on the same sets of code at different periods of time. It can also result in performance issues such as longer run times and potentially even vulnerabilities. |
| 1. Default Deny | Authorization and access into systems is an important component of security within an organization and a software program. If an unauthorized or unidentified individual is attempting to gain access into the system, then following the default deny method can prevent any intruders from infiltrating access into the system or other people’s accounts. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege states that an individual should not be given access/permission to features or information which they do not need to perform their given role or job. This assists in greatly reducing the overall number of individuals with privileged access which can become an issue in security. |
| 1. Sanitize Data Sent to Other Systems | Ensuring that data is properly sanitized before it’s sent to over systems can greatly assist in reducing the potential attacks such as buffer overflow and SQL injection. If something were to happen to the file during transit and an unauthorized individual got access, the sanitization would reduce the likelihood of tampering. |
| 1. Practice Defense in Depth | Implementing multiple layers of security is something which should be done in every single organization and software program. Having multiple layers to combat threats ensures that if one step is breached then there are still multiple others which must be broken into. Often times if there is an attack at one layer it can be found and contained before it gets to the next or further down the line. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques must be implemented at every stage in almost any manner. For software, this can entail code reviews and rigorous testing before implementation into the final product which in the end assists in improving the quality and security for both the user and company. This can also help developers catch security vulnerabilities at each stage which end up increasing overall security. |
| 1. Adopt a Secure Coding Standard | By following a set of secure coding principles developers can follow a guideline for their code and ensure its secure. Using tools such as OWASP and CERT can help a company create programs which are more secure and less likely to be broken into. |

### 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] | Do not write syntactically ambiguous declarations. |

**Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL53-CPP.+Do+not+write+syntactically+ambiguous+declarations**

| **Noncompliant Code** |
| --- |
| The code provided here is noncompliant primarily because the statement of unique lock is declaring m as an object and then constructing it. This results in the mutex object never actually becoming locked. |
| #include <mutex>    **static** std::mutex m;  **static** **int** shared\_resource;    **void** increment\_by\_42() {    std::unique\_lock<std::mutex>(m);    shared\_resource += 42;  } |

| **Compliant Code** |
| --- |
| This code is compliant because a lock object is assigned to the m variable which results in it actually becoming locked when it converts the constructor. |
| #include <mutex>    **static** std::mutex m;  **static** **int** shared\_resource;    **void** increment\_by\_42() {    std::unique\_lock<std::mutex> lock(m);    shared\_resource += 42;  } |

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

| **Principles(s):** Ambiguity Resolutions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.STRUCT.DECL.FNEST | Nested function declaration |
| LDRA tool suite | 9.7.1 | 296 S | Partially implemented |
| Helix QAC | 2024.1 | C++1109, C++2510 |  |
| Klockwork | 2024.1 | CERT.DLC.AMBIGUOUS\_DECL |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized memory. |

**Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory**

| **Noncompliant Code** |
| --- |
| The provided code is noncompliant because the integer I is uninitialized which results in the program returning an undefined behavior. |
| #include <iostream>    **void** f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| This code here is compliant because the I variable is assigned a value of zero. It is then initialized during the print statement and properly returned. |
| #include <iostream>    **void** f() {  **int** i = 0;    std::cout << i;  } |

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

| **Principles(s):** Compile cleanly at high warning levels. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Uninitialized-local-read  Uninitialized-variable-use | Fully Checked |
| CodeSonar | 8.1p0 | LANG.MEM.UVAR | Uninitialized variable |
| Coverity | 2017.07 | UNINIT | Implemented |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Fully implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CLG] | Do not attempt to modify string literals. |

**Source: https://wiki.sei.cmu.edu/confluence/display/c/STR30-C.+Do+not+attempt+to+modify+string+literals**

| **Noncompliant Code** |
| --- |
| This code is an example of noncompliance because the string literal is passed through the mkstemp function which then proceeds to change the actual string literal. |
| #include <stdlib.h>    **void** func(**void**) {    mkstemp("/tmp/edXXXXXX");  } |

| **Compliant Code** |
| --- |
| The provided code is an example of compliant code because instead of passing the string literal through a parameter it uses a named array in its place. |
| #include <stdlib.h>    **void** func(**void**) {  **static** **char** fname[] = "/tmp/edXXXXXX";    mkstemp(fname);  } |

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

| **Principles(s):** Do not cast away a const qualification. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | String-literal-modification  Write-to-string-literal | Fully Checked |
| Axivion Bauhaus suite | 7.2.0 | CertC-STR30 | Fully Implemented |
| Coverity | 2017.07 | PW | Deprecates conversion from a string literal to a char |
| LDRA tool suite | 9.7.1 | 157 S | Partially Implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL Injection. |

**Source: https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection**

| **Noncompliant Code** |
| --- |
| The provided code is an example of noncompliance because the username argument is unsanitized which could lead to a SQL injection attack. |
| **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 provided code below is an example of compliance because it places a question mark as a placeholder for the value within each argument. It also validates the length of the username. |
| **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):** Sanitize data passed to complex subsystems |
| --- |

**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 |
| Parasoft Jtest | 2023.1 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SpotBugs | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE  SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Detect and handle memory allocation errors. |

**Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors**

| **Noncompliant Code** |
| --- |
| We can see noncompliance because an array is created through the use of a function and the results of the value allocation are not checked. The statement itself can throw an exception if the value allocation ends up failing. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| This is compliant because it returns either a pointer or null pointer and if it returns the latter then the error is handled properly. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

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

| **Principles(s):** Detect and handle standard library errors |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| LDRA tool suite | 9.7.1 | 45 D | Partially Implemented |
| Polyspace Bug Finder | R2023b | CERT C++ : MEM52-CPP | Checks for unprotected dynamic memory allocation |
| PVS-Studio | 7.30 | V522, V668 |  |

#### 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. |

**Source: https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression**

| **Noncompliant Code** |
| --- |
| This is an example of noncompliance because without a static assert function directly into the targeted code we won’t be able to diagnose the issue unless it’s after the code has already been ran. |
| #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** |
| --- |
| We can see compliance here because the program uses a static assert function with can find errors at runtime rather than having to discover the error elsewhere. |
| #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 a static assertion to test the value of a constant expression. |
| --- |

**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 |
| CodeSonar | 8.1p0 | Customization | Users can implement a custom check that reports uses of the assert macro |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully Implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Honor exception specifications. |

**Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR55-CPP.+Honor+exception+specifications**

| **Noncompliant Code** |
| --- |
| We can see noncompliance here because it contains a no except statement which it means that it is no throwing, but the original statement itself can throw an exception depending on the memory. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {    v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| There is compliance within this code because it does not contain a no except function meaning that it ends up allowing all of the exceptions. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) {    v.resize(s); // May throw, but that is okay  } |

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

| **Principles(s):** Do not abruptly terminate the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Unhandled-throw-noexcept | Partially checked |
| CodeSonar | 81.p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| LDRA tool suite | 9.7.1 | 56 D | Partially implemented |
| RuleChecker | 22.10 | Unhandled-throw-noexcept | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory | [STD-008-CPP] | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| Within this code, the integer s has no longer referenced after it had been deallocated from its current value. If this remains, there is room for a potential vulnerability attack where code is inserted into the value of s. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| This code here is compliant because the memory is only deallocated when it is no longer needed. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** s;  } |

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

| **Principles(s):** Undefined behavior that is not covered by use after free errors |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 81.p0 | ALLOC.UAF | Use after free |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| LDRA tool suite | 9.7.1 | 483 S, 484 S | Partially Implemented |
| Polyspace Bug Finer | R2023b | CERT C++ : MEM50-CPP | Pointer access out of bounds |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Type | [STD-009-CLG] | Ensure that operations on signed integers do not result in overflow. |

**Source: https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow**

| **Noncompliant Code** |
| --- |
| Because of the signed operands, we have receive an overflow issue because of the addition that occurs between them. |
| **void** func(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum = si\_a + si\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This is compliance because it make sures the addition operation can not overflow at all. |
| #include <limits.h>    **void** f(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum;  **if** (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||        ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {      /\* Handle error \*/    } **else** {      sum = si\_a + si\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** Signed integer overflows that lead to buffer overflows. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Integer-overflow | Fully Checked |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially Implemented |
| TrustInSoft Analyzer | 1.38 | SIGNED\_OVERFLOW | Exhaustively verified |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | [STD-010-CPP] | Do not abruptly terminate the program. |

**Source: https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR50-CPP.+Do+not+abruptly+terminate+the+program**

| **Noncompliant Code** |
| --- |
| This code is noncompliant because it might cause the out statement to throw an exception. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| This code is compliant because our f class ends up taking control of any and all exceptions which are thrown by the throwing function class and do not throw. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.  **try** {      throwing\_func();    } **catch** (...) {      // Handle error    }  }  **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

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

| **Principles(s):** Improper check for unusual or exceptional conditions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Stdlib-use | Partially Checked |
| CodeSonar | 8.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| RuleChecker | 22.10 | Stdlib-use | Partially Checked |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

The DevSecOps diagram provided to us by Green Pace already contains a vast majority of features which must be included in an existing system. The pre production and production phases combine perfectly to create an infinite loop which can be followed for the future implementation and creation of features or products within the company. If there were any modifications that I were to make, it would most likely be at the build phase, as well as the transition and health check. If we look at the build phase, prioritizing the mindset of creating secure code and reviewing this before proceeding to the verify and test phase can save developers a ton of time on both bugs as well as potential security fixes in the future. For the transition and health check phase, we need to ensure that when we’re deploying security settings they align with our current systems and won’t cause issues in the future.

### 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 | Medium | Low (2) | 2 |
| STD-002-CPP | High | Probable | Medium | High (12) | 1 |
| STD-003-CLG | Low | Likely | Low | High (9) | 2 |
| STD-004-CPP | High | Likely | Medium | High (18) | 1 |
| STD-005-CPP | High | Likely | Medium | High (18) | 1 |
| STD-006-CLG | Low | Unlikely | High | Low (1) | 3 |
| STD-007-CPP | Low | Likely | Low | High (9) | 2 |
| STD-008-CPP | High | Likely | Medium | High (18) | 1 |
| STD-009-CLG | High | Likely | High | High (9) | 2 |
| STD-010-CPP | Low | Probable | Medium | Low (4) | 3 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest basically refers to the encryption of data while it’s currently stored. Since the data is not moving or being used, then it’s at rest which explains why the encryption is at rest. |
| Encryption in flight | Encryption in flight is the process of data being transported or transmitted over a network and between devices. Implementing this is vital as it helps reduce the potential attacks of data mid transport. |
| Encryption in use | Encryption in use is the process of protecting data which is in active use. This means when it’s being used or processed by an application. Again, this is an extremely important aspect to implement as it can prevent data being stolen during it’s interaction with applications or users. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of confirmed the identity of an individual when they’re looking to gain access to a system. The policy does apply to our system as we need to correctly authorize individuals so that they can gain access to our systems. |
| Authorization | Authorization is the next step because once a user is authenticated for who they really are, they must be authorized for their respective systems. For example, if our policy implements a role based access control, it’s vital that the user is correctly able to access only the files that they need to accomplish their jobs. |
| Accounting | Accounting is the process of auditing and logging the users who interact with the systems. This goes a little bit further than that as well as by collecting system events and information in the attempt to back track and find information about the system. This can assist in finding out about past or potential attacks to an existing system. |

**\***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 | 03/14/24 | Template Filling | Saram Nadeem |  |
| 1.2 | 04/14/24 | Project One | Saram Nadeem |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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