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



# Green Pace Secure Development Policy

# Kateryna Burkova

## 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 | To prevent major software vulnerabilities, we need to make sure to check the input data from untrusted sources. We can use external data sources, but the code needs to be checked for arguments and duplicates. A lot of applications are using external data sources and replicating it on their end. |
| 1. Heed Compiler Warnings | We need to use static and dynamic analysis tools. Complier warnings level should be the highest to warn us about any security violations. |
| 1. Architect and Design for Security Policies | We need to make sure while making the design of the code to follow security policies. Such as authentication, authorization, risk management etc. |
| 1. Keep It Simple | The code should not be overwhelmed with lines of unnecessary loops. The design of the code should remain simple and not complex. Complex codes are more vulnerable o fail and to run into errors. |
| 1. Default Deny | We need to have standards for the access to the application. There must be a set of rules for the users to have access. Users without this permission should be denied by default. |
| 1. Adhere to the Principle of Least Privilege | We need the principle of least privilege to reduce the risk of attacks. We don’t want our data to get compromised and customer information to be stolen because of this. We need to have elevated permissions for the authorized users to complete the specific task. |
| 1. Sanitize Data Sent to Other Systems | Sanitize data sent to other systems. We need to perform sanitization to dissuade injection attacks from attackers. |
| 1. Practice Defense in Depth | Using multiple layers of security preventing security risks from different layers. It is always good to have multiple security platforms to protect our clients. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance can increase security protection for the software. Using the most effective quality techniques will help us to protect our clients. We should have multiple testing phases, penetration testing and other tests performed on the product. |
| 1. Adopt a Secure Coding Standard | We should always adopt for the situation to use secure coding standard for development. |

### 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 modify the standard namespaces |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the declaration of x is added to the namespace std, resulting in undefined behavior. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| This compliant solution assumes the intention of the programmer was to place the declaration of x into a namespace to prevent collisions with other global identifiers. Instead of placing the declaration into the namespace std, the declaration is placed into a namespace without a reserved name. |
| namespace nonstd {  int x;  } |

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

| **Principles(s):** Validate input data. To prevent major software vulnerabilities, we need to make sure to check the input data from untrusted sources. We can use external data sources, but the code needs to be checked for arguments and duplicates. A lot of applications are using external data sources and replicating it on their end. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL58** |  |
| CodeSonar | 7.2p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| Parasoft C/C++test | 2022.2 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |
| Polyspace Bug Finder | R2022b | CERT C++: DCL58-CPP | Checks for modification of standard namespaces (rule fully covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not access an object outside of its lifetime |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a pointer to an object is implicitly converted to a virtual base class after the object's lifetime has ended, resulting in undefined behavior. |
| **struct** B {};    **struct** D1 : **virtual** B {};  **struct** D2 : **virtual** B {};    **struct** S : D1, D2 {};    **void** f(**const** B \*b) {}    **void** g() {    S \*s = **new** S;    // Use s  **delete** s;      f(s);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the lifetime of s is extended to cover the call to f(). |
| **struct** B {};    **struct** D1 : **virtual** B {};  **struct** D2 : **virtual** B {};    **struct** S : D1, D2 {};    **void** f(**const** B \*b) {}    **void** g() {    S \*s = **new** S;    // Use s    f(s);    **delete** s;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques. Quality assurance can increase security protection for the software. Using the most effective quality techniques will help us to protect our clients. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 42 D, 53 D, 77 D, 1 J, 71 S, 565 S | Partially implemented |
| RuleChecker | 22.10 | return-reference-local | Partially checked |
| Astrée | 22.10 | return-reference-local  dangling\_pointer\_use | Partially checked |
| Clang | 3.9 | -Wdangling-initializer-list | Catches some lifetime issues related to incorrect use of std::initializer\_list<> |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs). |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| [In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #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):** ValidateInput Data. To prevent major software vulnerabilities, we need to make sure to check the input data from untrusted sources. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| STR51-CPP | High | Likely | P18 | **L1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | assert\_failure |  |
| CodeSonar | 7.2p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Parasoft C/C++test | 2022.2 | [CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace Bug Finder | R2022b | CERT C++: STR51-CPP | Checks for string operations on null pointer (rule fully covered) |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-LLL | Do not use std::rand() for generating pseudorandom numbers |

| **Noncompliant Code** |
| --- |
| The following noncompliant code generates an ID with a numeric part produced by calling the rand() function. The IDs produced are predictable and have limited randomness. Further, depending on the value of RAND\_MAX, the resulting value can have modulo bias. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    id += std::to\_string(std::**rand**() % 10000);    // ...  } |

| **Compliant Code** |
| --- |
| The C++ standard library provides mechanisms for fine-grained control over pseudorandom number generation. It breaks random number generation into two parts: one is the algorithm responsible for providing random values (the engine), and the other is responsible for distribution of the random values via a density function (the distribution). The distribution object is not strictly required, but it works to ensure that values are properly distributed within a given range instead of improperly distributed due to bias issues. This compliant solution uses the Mersenne Twister algorithm as the engine for generating random values and a uniform distribution to negate the modulo bias from the noncompliant code example. |
| #include <random>  #include <string>    **void** f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    std::uniform\_int\_distribution<**int**> distribution(0, 10000);    std::random\_device rd;    std::mt19937 engine(rd());    id += std::to\_string(distribution(engine));    // ...  } |

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

| **Principles(s):** Architect and Design for Security Policies. We need to make sure while making the design of the code to follow security policies. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | bad-function (AUTOSAR.26.5.1A) | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MSC50 |  |
| LDRA tool suite | [9.7.1 | 44 S | Enhanced Enforcement |
| RuleChecker | 22.10 | bad-function (AUTOSAR.26.5.1A) | Fully checked |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-LLL] | Properly deallocate dynamically allocated resources |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the local variable space is passed as the expression to the placement new operator. The resulting pointer of that call is then passed to ::operator delete(), resulting in undefined behavior due to ::operator delete() attempting to free memory that was not returned by ::operator new(). |
| #include <iostream>    **struct** S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {    alignas(**struct** S) **char** space[**sizeof**(**struct** S)];    S \*s1 = **new** (&space) S;      // ...    **delete** s1;  } |

| **Compliant Code** |
| --- |
| This compliant solution removes the call to ::operator delete(), instead explicitly calling s1's destructor. This is one of the few times when explicitly invoking a destructor is warranted. |
| #include <iostream>    **struct** S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {    alignas(**struct** S) **char** space[**sizeof**(**struct** S)];    S \*s1 = **new** (&space) S;      // ...      s1->~S();  } |

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

| **Principles(s):** Adopt a Secure Coding Standard We should always adopt for the situation to use secure coding standard for development. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM51 |  |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | Partially implemented |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete | Checked by clang-tidy, but does not catch all violations of this rule |
| SonarQube C/C++ Plugin | 4.10 | S1232 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-LLL] | Do not access a closed file |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the stdout stream is used after it is closed: |
| #include <stdio.h>    **int** close\_stdout(**void**) {  **if** (**fclose**(stdout) == EOF) {  **return** -1;    }    **printf**("stdout successfully closed.\n");  **return** 0;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, stdout is not used again after it is closed. This must remain true for the remainder of the program, or stdout must be assigned the address of an open file object. |
| #include <stdio.h>    **int** close\_stdout(**void**) {  **if** (**fclose**(stdout) == EOF) {  **return** -1;    }    **fputs**("stdout successfully closed.", stderr);  **return** 0;  } |

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

| **Principles(s):** ValidateInput Data. To prevent major software vulnerabilities, we need to make sure to check the input data from untrusted sources. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 |  | Supported |
| LDRA tool suite | 9.7.1 | 48 D | Partially implemented |
| PRQA QA-C | 9.7 | 2696, 2697, 2698 |  |
| SonarQube C/C++ Plugin | 3.11 | S3588 |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-LLL] | Do not leak resources when handling exceptions |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, pst is not properly released when process\_item throws an exception, causing a resource leak. |
| #include <new>    **struct** SomeType {    SomeType() noexcept; // Performs nontrivial initialization.    ~SomeType(); // Performs nontrivial finalization.  **void** process\_item() noexcept(**false**);  };    **void** f() {    SomeType \*pst = **new** (std::**nothrow**) SomeType();  **if** (!pst) {      // Handle error  **return**;    }    **try** {      pst->process\_item();    } **catch** (...) {      // Process error, but do not recover from it; rethrow.  **throw**;    }  **delete** pst;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the exception handler frees pst by calling delete. |
| #include <new>    **struct** SomeType {    SomeType() noexcept; // Performs nontrivial initialization.    ~SomeType(); // Performs nontrivial finalization.    **void** process\_item() noexcept(**false**);  };    **void** f() {    SomeType \*pst = **new** (std::**nothrow**) SomeType();  **if** (!pst) {      // Handle error  **return**;    }  **try** {      pst->process\_item();    } **catch** (...) {      // Process error, but do not recover from it; rethrow.  **delete** pst;  **throw**;    }  **delete** pst;  } |

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

| **Principles(s):** Default Deny We need to have standards for the access to the application. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.2p0 | ALLOC.LEAK | Leak |
| LDRA tool suite | 9.7.1 | 50 D | Partially implemented |
| Helix QAC | 2022.4 | DF4756, DF4757, DF4758 |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-ERR57-a | Ensure resources are freed |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrency | [STD-008-LLL] | Prevent data races when accessing bit-fields from multiple threads |

| **Noncompliant Code** |
| --- |
| Adjacent bit-fields may be stored in a single memory location. Consequently, modifying adjacent bit-fields in different threads is undefined behavior, as shown in this noncompliant code example. |
| **struct** MultiThreadedFlags {    unsigned **int** flag1 : 2;    unsigned **int** flag2 : 2;  };    MultiThreadedFlags flags;    **void** thread1() {    flags.flag1 = 1;  }    **void** thread2() {    flags.flag2 = 2;  } |

| **Compliant Code** |
| --- |
| This compliant solution protects all accesses of the flags with a mutex, thereby preventing any data races. |
| #include <mutex>    **struct** MultiThreadedFlags {    unsigned **int** flag1 : 2;    unsigned **int** flag2 : 2;  };    **struct** MtfMutex {    MultiThreadedFlags s;    std::mutex mutex;  };    MtfMutex flags;    **void** thread1() {    std::lock\_guard<std::mutex> lk(flags.mutex);    flags.s.flag1 = 1;  }    **void** thread2() {    std::lock\_guard<std::mutex> lk(flags.mutex);    flags.s.flag2 = 2;  } |

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

| **Principles(s):** Heed Compiler Warnings. We need to use static and dynamic analysis tools. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | read\_write\_data\_race | Supported |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-CON52 |  |
| Coverity | 6.5 | RACE\_CONDITION | Fully implemented |
| PRQA QA-C++ | 4.4 | 1774, 1775 | Enforced by MTA |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | STD-009-LLL | Do not access a cv-qualified object through a cv-unqualified type |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the function g() is passed a const int &, which is then cast to an int & and modified. Because the referenced value was previously declared as const, the assignment operation results in undefined behavior. |
| **void** g(**const** **int** &ci) {  **int** &ir = **const\_cast**<**int** &>(ci);    ir = 42;  }    **void** f() {  **const** **int** i = 4;    g(i);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function g() is passed an int &, and the caller is required to pass an int that can be modified. |
| **void** g(**int** &i) {    i = 42;  }    **void** f() {  **int** i = 4;    g(i);  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems Sanitize data sent to other systems. We need to perform sanitization to dissuade injection attacks from attackers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | pointer-qualifier-cast-const | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-EXP55 |  |
| RuleChecker | 22.10 | pointer-qualifier-cast-const | Partially checked |
| SonarQube C/C++ Plugin | 4.10 | S859 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions of error handling | STD-010-LLL | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| [In this noncompliant code example, multiple numeric values are converted from the standard input stream. However, if the text received from the standard input stream cannot be converted into a numeric value that can be represented by an int, the resulting value stored into the variables i and j may be unexpected. |
| #include <iostream>    **void** f() {  **int** i, j;    std::cin >> i >> j;    // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, exceptions are enabled so that any conversion failure results in an exception being thrown. However, this approach cannot distinguish between which values are valid and which values are invalid and must assume that all values are invalid. Both the badbit and failbit flags are set to ensure that conversion errors as well as loss of integrity with the stream are treated as exceptions. |
| #include <iostream>    **void** f() {  **int** i, j;      std::cin.exceptions(std::istream::failbit | std::istream::badbit);  **try** {      std::cin >> i >> j;      // ...    } **catch** (std::istream::failure &E) {      // Handle error    }  } |

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

| **Principles(s):** Keep It Simple. The code should not be overwhelmed with lines of unnecessary loops. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR62 |  |
| Helix QAC | 2022.4 | C++3161 |  |
| Klocwork | 2022.4 | CERT.ERR.CONV.STR\_TO\_NUM |  |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-ERR62-a | The library functions atof, atoi and atol from library stdlib.h shall not be used |

### 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.

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-002-CPP | High | Probable | High | P6 | L2 |
| STD-003-CPP | High | Likely | High | P18 | L1 |
| STD-004-CPP | Medium | Unlikely | Low | P6 | L2 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-007-CPP | Low | Probable | High | P2 | L3 |
| STD-008-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-009-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is data that isn’t being used on a hard drive. If the attacker gets access to the hard drive the attacker must hack the data without the key. The encryption at rest also means that the data is not travelling between the devices. Encrypted data at rest can be saved to a hard drive or portable device. |
| Encryption at flight | Encryption at flight is data travelling from one point to another. This process encrypts data while it is being transmitted. Such as emails, messengers, calls. This type of data is usually less secure than data at rest, and it is an easy target for hackers. |
| Encryption in use | It is the data that is being actively used by the user with the encryption key. In this stage data is vulnerable. It can lead to a human error and expose the data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a process to verify user identity such as password and a second type of authentication can be a phone number with a text message. The most current is DUO or Microsoft multifactor authentication. User logins. |
| Authorization | Authorization for data access determines what groups user can be in. Write, read, full access permissions groups usually are getting authorized with the administrator. User level of access. Addition of new users |
| Accounting | Accounting is process to monitor users actions with the data. It usually keeps track of what user can or can’t access, and what changes were made to the file stream. Files accessed by users. Changes to the database. |

**\***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.

* 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 | 1/15/2023 | Policy | Kateryna Burkova |  |
| 1.2 | 1/28/2023 | Security Policy | Kateryna Burkova |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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