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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 14](#_Toc52464064)

[Coding Standard 7 16](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 20](#_Toc52464067)

[Coding Standard 10 22](#_Toc52464068)

[Defense-in-Depth Illustration 24](#_Toc52464069)

[Automation 24](#_Toc52464074)

[Summary of Risk Assessments 25](#_Toc52464075)

[Policies for Encryption and Triple A 26](#_Toc52464076)

[Audit Controls and Management 27](#_Toc52464078)

[Enforcement 27](#_Toc52464079)

[Exceptions Process 27](#_Toc52464080)

[Distribution 28](#_Toc52464081)

[Policy Change Control 28](#_Toc52464082)

[Policy Version History 28](#_Toc52464083)

[Appendix A Lookups 28](#_Toc52464084)

[Approved C/C++ Language Acronyms 28](#_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.

### 

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Input validation should exist within the code such that it recognizes when erroneous or malicious input is being read. Input that exceeds a character limit, presents as a different data type, or appears otherwise suspicious should be discarded and the user should be prompted to re-enter the input correctly. |
| 1. Heed Compiler Warnings | While errors will prevent code from compiling, warnings will not. It is important to recognize and rectify issues brought forward by compiler warnings to maintain secure code. Though code may compile as expected with warnings, failure to fix any known issues that are presented can result in a faulty and vulnerable product. |
| 1. Architect and Design for Security Policies | Architecture and design regard the systems and related sub-systems that may be present in software and the security measures that pertain to each one at a sufficient level. Different elements of a program may require different levels of privilege regarding who can access and edit these elements. |
| 1. Keep It Simple | Keeping code simple makes debugging and finding errors easier, as well as making the code understandable to other programmers. Concise and well-commented code makes both development and maintenance easier for those involved within these processes. |
| 1. Default Deny | Unless proper verification is provided, the default action of a program should be to deny access to actions or elements that are outside of the user’s privilege level. |
| 1. Adhere to the Principle of Least Privilege | Users should only be given permissions designated to their role, and the privilege level should be kept as low as possible. The program should be able to execute all necessary tasks for a user at the lowest privilege level without need for intervention of higher privilege levels. |
| 1. Sanitize Data Sent to Other Systems | Data sent to other systems should be “sanitized” so that the data is able to cross the trust boundary with the other system. This type of data should not include extraneous characters or function calls that suggest malicious data is being passed. |
| 1. Practice Defense in Depth | A multi-layered defense system should be used such that if one layer of defense fails or becomes vulnerable, there is another beneath it that is able to catch an attack. It is also worth keeping in mind that with each new project, the depth of number of layers may vary; it is important to know what is enough and what is too little or too much. |
| 1. Use Effective Quality Assurance Techniques | While there are several QA techniques that may and will be employed, it should be noted that one should always test early and test often. Consistently referring to the requirements of a product is also a key element to keeping up with QA. |
| 1. Adopt a Secure Coding Standard | Secure coding standards provide a guideline to writing code while preventing vulnerabilities. The main standard presented in this document follows the [SEI CERT C++ Coding Standard](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682). |

### C/C++ Ten Coding Standards

#### Coding Standard 1

| **Coding Standard** | **Label** | **The One-Definition Rule (ODR)** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | A C++ may include multiple translation units that will be linked and require the same object be referenced. This object should only be defined once. |

**source:** [DCL60-CPP. Obey the one-definition rule](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule)

| **Noncompliant Code** |
| --- |
| Two units define a class of the same name with differing definitions. They are not defined using the same sequence of tokens and result in undefined behavior. |
| // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |

| **Compliant Code** |
| --- |
| If the same class definition must be visible in both translation units because of common usage, the solution is to use a header file to introduce the object into both translation units. |
| // S.h  **struct** S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

| **Principles(s):** Keep It Simple – Defining two classes of the same name with differing definitions inherently complicates the code, making differentiation between the classes confusing. The use of header files keeps the code organized. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | Low (3) | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading | Partially checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.DEF.FDH LANG.STRUCT.DEF.ODH | Function defined in header file Object defined in header file |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| Polyspace Bug Finder | R2023b | CERT C++: DCL60-CPP | Checks for inline constraints not respected (rule partially covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do Not Read Uninitialized Memory** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Variables initialized without a value can assume unexpected values when read, resulting in undefined behavior. |

**source:** [EXP53-CPP. Do not read uninitialized memory](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory)

| **Noncompliant Code** |
| --- |
| An uninitialized variable is being evaluated to print its value, which results in undefined behavior. |
| #include <iostream>    **void** f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| The variable is initialized with a value prior to printing, avoiding undefined behavior. |
| #include <iostream>    **void** f() {  **int** i = 0;    std::cout << i;  } |

| **Principles(s):** Heed Compiler Warnings – Use of an uninitialized variable will cause an IDE such as Visual Studio to present an error of code C6001 to the developer reading “Using uninitialized memory ‘variableName’”. This warning should be regarded to avoid situations where uninitialized variables are attempted to be evaluated. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High (12) | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | uninitialized-read | Partially checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.RPL LANG.MEM.UVAR | Return pointer to local Uninitialized variable |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| Polyspace Bug Finder | R2023b | CERT C++: EXP53-CPP | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that Storage for Strings has Sufficient Space for Character Data** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Sending data to a buffer that is not large enough to hold the data can result in buffer overflow. |

**source:** [STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator)

| **Noncompliant Code** |
| --- |
| The resulting input is unbounded, which can result in buffer overflow. |
| #include <iostream>    **void** f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| std::string should be used instead of a bounded array in order to ensure that data is not truncated and that buffer overflow will not occur. |
| #include <iostream>  #include <string>    **void** f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

| **Principles(s):** Validate Input Data – When initializing a variable meant to take user input, it’s important to recognize when declarations may not allow for certain input types or lengths. In this noncompliant code example, using limited storage such as **char** buf[12]; can cause buffer overflow if not handled correctly.  Keep It Simple – More often than not, it’s safer and more efficient to use a std::string object rather than a char array as you don’t need to worry about properly allocating or deallocating memory with a string object. char may only be necessary in specific instances where memory management and performance are crucial. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High (18) | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 8.1p0 | MISC.MEM.NTERM  LANG.MEM.BO LANG.MEM.TO | No space for null terminator  Buffer overrun Type overrun |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-STR50-b CERT\_CPP-STR50-c CERT\_CPP-STR50-e CERT\_CPP-STR50-f CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| Polyspace Bug Finder | R2023b | 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. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Malicious users can attempt to parse unwanted data to cause SQL to behave unexpectedly; measures should be taken to prevent these types of attacks. |

**source:** [IDS00-J. Prevent SQL injection](https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection) (Java version used as reference)

| **Noncompliant Code** |
| --- |
| Non-compliant code does not properly validate or sanitize the information. |
| 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(); |

| **Compliant Code** |
| --- |
| Compliant code checks for username length in order to avoid malicious data being sent to the query and PreparedStatement stmt uses setString() for more secure validation. |
| 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(); |

| **Principles(s):** Validate Input Data – In instances where variables that intake user input are being used for SQL processes, proper validation methods should be set to detect when an injection attempt could be present.  Default Deny – In an instance where an SQL injection is detected, default behavior should deny the user access to any further action until the possibility of attack is mitigated.  Sanitize Data Sent to Other Systems – If any data regarding SQL processes is being transported to other systems, one must ensure that any attempts of an injection are detected and removed, or otherwise default to denying the transport of data until it is deemed safe. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High (18) | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| CodeSonar | 8.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Fortify | 1.0 | HTTP\_Response\_Splitting SQL\_Injection\_\_Persistence SQL\_Injection | Implemented |
| Parasoft Jtest | 2023.1 | CERT.IDS00.TDSQL | Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do Not Access Freed Memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Attempting to access or reference already-freed memory can result in unwanted behaviors or vulnerabilities where arbitrary code can be run. |

**source:** [MEM50-CPP. Do not access freed memory](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory)

| **Noncompliant Code** |
| --- |
| Variable is dereferenced after deletion, allowing for arbitrary code to be run with the permissions of the vulnerable process. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| Variable is not deleted until it is no longer needed. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** s;  } |

| **Principles(s):** Heed Compiler Warnings – The evaluation of a deleted object will present the developer with the same error as if an uninitialized variable were being evaluated. Developers must heed this warning and be sure to rectify the deletion’s placement within the code to avoid undefined behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High (18) | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | dangling\_pointer\_use | n/a |
| CodeSonar | 8.1p0 | ALLOC.UAF | Use after free |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Polyspace Bug Finder | R2023b | CERT C++: MEM50-CPP | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Understand the Termination Behavior of assert()** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | assert() puts diagnostics into programs, expanding to a void expression. If *expression* in assert(*expression*) is false, an abort will be called; aborting will not occur is *expression* is not false. It is important to be aware of this when a programmer intends for the program to abort at a specific moment. |

**source:** [ERR06-C. Understand the termination behavior of assert() and abort()](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152296)

| **Noncompliant Code** |
| --- |
| Code shows an example where a function is called before the program exits to cleanup. **assert**() detects nothing wrong, so program does not abort. |
| **void** cleanup(**void**) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    **int** main(**void**) {  **if** (**atexit**(cleanup) != 0) {      /\* Handle error \*/    }    /\* ... \*/  **assert**(/\* Something bad didn't happen \*/);  } |

| **Compliant Code** |
| --- |
| In this compliant example, **if** () is used instead of **assert**() in order to detect an error and abort the program appropriately. |
| **void** cleanup(**void**) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    **int** main(**void**) {  **if** (**atexit**(cleanup) != 0) {      /\* Handle error \*/    }    /\* ... \*/  **if** (/\* Something bad happened \*/) {  **exit**(EXIT\_FAILURE);    }  } |

| **Principles(s):** Use Effective Quality Assurance Techniques – Assert and abort functions are typical QA techniques applied to early development of software for testing. As such, it’s important to understand the behaviors of their use in order to QA test effectively. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low (4) | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE | n/a | n/a | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced enforcement |
| Parasoft C/C++test | 2023.1 | CERT\_C-ERR06-a | Do not use assertions |
| PC-lint Plus | 1.4 | 586 | Fully supported |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle All Exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | All exceptions thrown by a program require a matching exception handler. Failure to provide matching handlers can cause uncontrolled termination of the program. |

**source:** [ERR51-CPP. Handle all exceptions](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions)

| **Noncompliant Code** |
| --- |
| Neither f() normain() catch exceptions thrown by throwing\_func(); lack of a matching handler causes the program to terminate. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| main() entry point handles all exceptions which allows for appropriate management of external resources. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {  **try** {      f();    } **catch** (...) {      // Handle error    }  } |

| **Principles(s):** Heed Compiler Warnings – Catching exceptions allows the compiler to unwind safely; as such, exceptions are often related to how the compiler functions. Leaving uncaught exceptions will cause unexpected program termination, so it is import to make sure all exceptions are caught before compiling. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | Low (4) | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | main-function-catch-all early-catch-all | Partially checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR51-a CERT\_CPP-ERR51-b | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| Polyspace Bug Finder | R2023b | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do Not Delete a Polymorphic Object Without a Virtual Destructor** |
| --- | --- | --- |
| **Object Oriented Programming (OOP)** | [STD-008-CPP] | Deleting an object of a derived class without using a virtual destructor can result in undefined behavior. |

**source:** [OOP52-CPP. Do not delete a polymorphic object without a virtual destructor](https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP52-CPP.+Do+not+delete+a+polymorphic+object+without+a+virtual+destructor)

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

| **Compliant Code** |
| --- |
| Destructor for Base {} is explicitly declared, ensuring that polymorphic deletion functions as intended. |
| **struct** Base {  **virtual** ~Base() = **default**;  **virtual** **void** f();  };    **struct** Derived : Base {};    **void** f() {    Base \*b = **new** Derived();    // ...  **delete** b;  } |

| **Principles(s):** Adopt a Secure Coding Standard – The use of a virtual destructor for polymorphic deletion is a standard within secure coding. Having secure coding standards ensures that such scenarios are made aware to developers so that they may avoid running into problems associated with these types of objects. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | Medium (9) | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | non-virtual-public-destructor-in-non-final-class | Partially checked |
| CodeSonar | 8.1p0 | LANG.STRUCT.DNVD | delete with Non-Virtual Destructor |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-OOP52-a | Define a virtual destructor in classes used as base classes which have virtual functions |
| Polyspace Bug Finder | R2023b | [Insert text.] | Checks for situations when a class has virtual functions but not a virtual destructor (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Close Files When They Are No Longer Needed** |
| --- | --- | --- |
| **File Handling** | [STD-009-CPP] | An open file must be closed once it is no longer needed within the program, and especially be made closed before program termination. |

**source:** [FIO51-CPP. Close files when they are no longer needed](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed)

| **Noncompliant Code** |
| --- |
| The file object is not called to close before the program terminates, and the terminate() call does not call destructors that will close the file otherwise. |
| #include <exception>  #include <fstream>  #include <string>  **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    std::terminate();  } |

| **Compliant Code** |
| --- |
| Code ensures file resources are properly closed before the terminate() call. |
| #include <exception>  #include <fstream>  #include <string>  **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    file.close();  **if** (file.fail()) {      // Handle error    }    std::terminate();  } |

| **Principles(s):** Adopt a Secure Coding Standard – The act of closing a file within a software’s code when it is no longer needed is an industry standard that ensures that the file can be accessed by other processes once it is no longer needed by the program. Failing to close the file will result in it being held by the program and other processes will not be able to acquire the file until the program is terminated. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low (4) | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2024.1 | DF4786, DF4787, DF4788 | n/a |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Polyspace Bug Finder | R2023b | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Guarantee that Container Indices and Iterators are Within the Valid Range** |
| --- | --- | --- |
| **Containers** | [STD-010-CPP] | Inappropriate use of iterators and/or indices can result in writing outside the bounds of a table and result in undefined behavior. |

**source:** [CTR50-CPP. Guarantee that container indices and iterators are within the valid range](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR50-CPP.+Guarantee+that+container+indices+and+iterators+are+within+the+valid+range)

| **Noncompliant Code** |
| --- |
| pos is declared as an *int* variable which can take on a negative value, possibly resulting in a write outside the bounds of table. |
| #include <cstddef>    **void** insert\_in\_table(**int** \*table,  std::**size\_t** tableSize, **int** pos, **int** value) {  **if** (pos >= tableSize) {      // Handle error  **return**;    }    table[pos] = value;  } |

| **Compliant Code** |
| --- |
| pos being declared as *size\_t* prevents the passing of negative arguments. |
| #include <cstddef>  **void** insert\_in\_table(**int** \*table, std::**size\_t** tableSize,  std::**size\_t** pos, **int** value) {  **if** (pos >= tableSize) {      // Handle error  **return**;    }    table[pos] = value;  } |

| **Principles(s):** Adopt a Secure Coding Standard – Reaching outside the bounds of a container can be a common mistake in development. With a secure coding standard, developers can develop an efficient way to prevent this type of occurrence from happening. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | Medium (9) | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **overflow\_upon\_dereference** | n/a |
| CodeSonar | 8.1p0 | **LANG.MEM.BO LANG.MEM.BU LANG.MEM.TO LANG.MEM.TU LANG.MEM.TBA LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.STRUCT.PARITH** | Buffer overrun Buffer underrun Type overrun Type underrun Tainted buffer access Pointer before beginning of object Pointer past end of object Pointer Arithmetic |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-CTR50-a** | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2023b | CERT C++: CTR50-CPP | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

### Defense-in-Depth Illustration

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



### Automation



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 DevOps policies currently provided by Green Pace are a great starting point for adding security measures that make a functioning DevSecOps platform possible. As one would imagine, the addition of security is what separates DevSecOps from DevOps, and it’s expected that automation methods will be integrated into as many stages of the current DevOps development cycle (portrayed in the above image) as possible.

In the *Assess and plan* stage, one can expect to now consider the typical attack methods and patterns that take place within systems and applications, as well as brainstorming countermeasures to predict and defend against these types of attacks. For the *Design* and *Build* stages, one can expect to reference this security policy and adopt the secure coding practices provided to create a more secure product that is less susceptible to attempts at malicious activity. In the *Verify and test* stage, automated unit testing will be implemented to verify the consistency and security of individual aspects of an application. Integration testing will then take place where the individual units are integrated into a whole application, where one can then test the product’s defense against traditional hacking methods, such as brute forcing or SQL injection.

In the *Production* phase, one will want to place more focus on protecting the underlying OS before a release is considered, and this especially ties into the Principle of Least Privilege as listed in the **Ten Core Security Principles**. Automated network monitoring and performance logs should be implemented to notify of suspicious network traffic and user activities as production moves towards the *Maintain and stabilize* stage.

### Summary of Risk Assessments

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | High | Low (3) | 3 |
| STD-002-CPP | High | Probable | Medium | High (12) | 1 |
| STD-003-CPP | High | Likely | Medium | High (18) | 1 |
| STD-004-CPP | High | Likely | Medium | High (18) | 1 |
| STD-005-CPP | High | Likely | Medium | High (18) | 1 |
| STD-006-CPP | Medium | Unlikely | Medium | Low (4) | 3 |
| STD-007-CPP | Low | Probable | Medium | Low (4) | 3 |
| STD-008-CPP | Low | Likely | Low | Medium (9) | 2 |
| STD-009-CPP | Medium | Unlikely | Medium | Low (4) | 3 |
| STD-010-CPP | High | Likely | High | Medium (9) | 2 |

### Policies for Encryption and Triple A

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Encryption at rest regards the encryption of data when held in a storage device such as an SSD, HDD, USB drive, CD, or any other storage medium. By implementing disk encryption, sensitive data can be protected when stored on a disk device, whether internal or external, as an attacker would require the necessary keys to access the data. |
| Encryption in flight | Encryption in flight regards data that is being transmitted between two or more different devices or systems. To protect sensitive data that is being transmitted, it’s important to have a secure connection protocol that encrypts the data leaving it inaccessible to attackers. The use of protocols such as Virtual Private Networks (VPNs), Secure Socket Layers (SSLs), Transport Layer Security (TSL), or Hypertext Transfer Protocol Secure (HTTPS) are some notable examples. |
| Encryption in use | Encryption in use regards data that is created by, edited by, or otherwise in-use by a user. Utilizing protocols like a .NET protected memory class and homomorphic encryption can protect sensitive data in use from being accessible to attackers. Homomorphic encryption in particular applies a conversion to data resulting in ciphertext that can be edited and manipulated as if it were plaintext in order to keep the data secure. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a process in which a program or server confirms that the user is who they say they are, typically by providing a pre-registered username or email along with a password. In general, the input of a correct password tied to the registered username/email is the discerning factor of whether the user should be given access to the profile associated with the website or application. 2-Factor Authentication (2FA) is another authentication method that ties a user’s login information to a separate device such as a cell phone. With 2FA, logging in with an appropriate username/email and password combination will trigger an SMS message sending to the user’s phone with a unique code; this code would be required to complete the authentication process and allow the user access. |
| Authorization | Authorization occurs after authentication and governs the user’s level of access within the associated platform. This directly ties to the Deny by Default principle, as a user’s authorization level will allow certain access but default to denying them to actions outside of their privilege level. The purpose of these roles is to leave specific data and actions out of reach to unauthorized users. For instance, you would want a typical user to have access to their profile information and the ability to change it when needed, but you would not want such a user to have access to administrative functions such as the addition/deletion of users or the ability to read confidential files. |
| Accounting | Accounting is the process of tracking any changes made to data within a database, server, or other type of system. Accounting tracks what files are accessed by which users as well as any editing or additions/deletions are made to a system. The process of accounting can also generate reports of this data, which can be beneficial to pinpointing the cause of a data breach that is discovered after the fact. |

## 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 | 03/24/2024 | Module 3 Milestone | Michael Duteau | [Insert text.] |
| 1.2 | 04/14/2024 | Project 1 | Michael Duteau | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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