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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 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 19](#_Toc52464066)

[Coding Standard 9 21](#_Toc52464067)

[Coding Standard 10 23](#_Toc52464068)

[Defense-in-Depth Illustration 25](#_Toc52464069)

[Project One 25](#_Toc52464070)

[1. Revise the C/C++ Standards 25](#_Toc52464071)

[2. Risk Assessment 25](#_Toc52464072)

[3. Automated Detection 25](#_Toc52464073)

[4. Automation 25](#_Toc52464074)

[5. Summary of Risk Assessments 26](#_Toc52464075)

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

[7. Map the Principles 27](#_Toc52464077)

[Audit Controls and Management 28](#_Toc52464078)

[Enforcement 28](#_Toc52464079)

[Exceptions Process 28](#_Toc52464080)

[Distribution 29](#_Toc52464081)

[Policy Change Control 29](#_Toc52464082)

[Policy Version History 29](#_Toc52464083)

[Appendix A Lookups 29](#_Toc52464084)

[Approved C/C++ Language Acronyms 29](#_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 | Validate input from all data sources. This eliminates software vulnerabilities when dealing with untrusted data sources like network interfaces, personal files, and environmental variables. |
| 1. Heed Compiler Warnings | Compile code frequently while using the highest warning level available for the compiler that you are using. If you get a warning, be sure to eliminate such warning by altering the code and recompiling to ensure that the warning has been eliminated. |
| 1. Architect and Design for Security Policies | Design your software to implement and enforce security policies. For example, if certain functionalities of your software will require different privileges at different intervals, design the software to be divided into the subsystems. The software will then be able to run without overriding privilege policies. |
| 1. Keep It Simple | Complex designs can often lead to errors during the implementation phase or software vulnerabilities. So, when designing software, keep the design as simple as possible. |
| 1. Default Deny | When making software decisions about access, make the decision based on permission. Meaning that the default will always be that access is denied unless certain conditions are met based on permission criteria. |
| 1. Adhere to the Principle of Least Privilege | Any process that requires elevated permission should take the least amount of time and should be accessed for the least amount of time. This will reduce the opportunities that an attacker has to access privileged information. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data that may be passed on to a complex subsystem. The act of sanitizing data includes purposefully deleting or destroying data to make sure that it can’t be recovered. |
| 1. Practice Defense in Depth | Implement multiple defense strategies so that if one fails, another can prevent a security flaw and patch up the vulnerability. |
| 1. Use Effective Quality Assurance Techniques | Incorporate different testing methods as quality assurance methods. Also incorporate individual security reviews as they can bring a different perspective in identifying security vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Apply secure coding standards for the coding languages and platforms that are being used to develop your software. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Never qualify a reference type with const or volatile** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Never qualify a reference type with const or volatile. An attempt to qualify a reference type as const or volatile will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this example, a const-qualified reference to a character is formed instead of a reference to a const-qualified character. |
| #include <iostream>    **void** function(**char** c) {  **char** &**const** a = c;    a = 'a';    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| The compliant solution correctly identifies ‘a’ as a reference to a const-qualified character. |
| #include <iostream>    **void** function(**char** c) {  **const** **char** &a = c;    a = 'p';    std::cout << c << std::endl;  } |

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

| **Principle(s):** The principle **Validate Input Data** would apply here. Although the data is not being entered by a user or by the system, an incorrectly qualified reference could result in a vulnerability as the behavior of the variable would be undefined. It’s just as important to verify references as it is to verify variable values. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **0014** | Automatically applies the C++ guidelines |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | N/A | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Declaring a variable in a switch case** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not declare a variable within a switch statement before the first use case. If a programmer tries to declare variables in the switch case and then use them within any of the case statements, the variables will contain indeterminate values. |

| **Noncompliant Code** |
| --- |
| This noncompliant example declares variables within the switch statement before the first use case: |
| #include <stdio.h>    **extern** **void** function(**int** i);    **void** f(**int** expr) {  **switch** (expr) {  **int** i = 5;      function(i);  **case** 0:      i = 16;  **default**:  **printf**("%d\n", i);    }  } |

| **Compliant Code** |
| --- |
| In the compliant solution, the statements before the first use case occur before the switch statement: |
| #include <stdio.h>    **extern** **void** function(**int** i);    **int** f(**int** expr) {  **int** i = 5;    function(i);    **switch** (expr) {  **case** 0:        i = 16;  **default**:  **printf**("%d\n", i);    }  **return** 0;  } |

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

| **Principles(s):** The principle **Validate Input Data** would apply here. Although the data is not being entered by a user or by the system, an incorrectly declared variable could result in a vulnerability as the value of the variable would be undefined. . |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **385 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | **CERT\_C-DCL41-a** | A switch statement shall only contain switch labels and switch clauses, and no other code |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **527** | Assistance provided |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C: Rule DCL41-C](https://www.mathworks.com/help/bugfinder/ref/certcruledcl41c.html) | Checks for ill-formed switch statements (rule partially covered) |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Reset string on fgets() and fgetws()** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Reset strings on fgets() and fgetws() failure. If either of the functions fails, the contents will be indeterminate. Therefore we need to reset the string to a known value to avoid errors on subsequent string manipulation functions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, buf is not reset and has indeterminate contents. |
| #include <stdio.h>    **enum** { BUFFER\_SIZE = 1024 };  **void** function(**FILE** \*file) {  **char** buf[BUFFER\_SIZE];    **if** (**fgets**(buf, **sizeof**(buf), file) == NULL) {    }  } |

| **Compliant Code** |
| --- |
| In this compliant example, buf is et to an empty string in the case that fgets() fails. |
| #include <stdio.h>    **enum** { BUFFER\_SIZE = 1024 };    **void** function(**FILE** \*file) {  **char** buf[BUFFER\_SIZE];    **if** (**fgets**(buf, **sizeof**(buf), file) == NULL) {      \*buf = '\0';    }  } |

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

| **Principles(s):** The principle **Validate Input Data** would apply here. Although the data is not being entered by a user or by the system, refusing to reset a string upon failure would result in an indeterminate value that could then be passed on throughout the program exposing a vulnerability within the system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | L4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **LANG.MEM.UVAR** | Uninitialized Variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2022.3 | **C4861, C4862, C4863**  **C++4861, C++4862, C++4863** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | **CERT\_C-FIO40-a** | Reset strings on fgets() or fgetws() failure |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Preventing SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | STD-001-JAV | Prevent SQL Injection by using parameterized queries and stored procedures. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the programmer uses a prepared statement to avoid a SQL Injection. However, the unsanitized variable username is incorporated into the prepared statement allowing a SQL Injection. |
| **public** **void** doPrivilegedAction(      String username, **char**[] password    ) **throws** SQLException {      Connection connection = getConnection();  **if** (connection == **null**){      }  **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");        } |

| **Compliant Code** |
| --- |
| In the compliant solution, the programmer uses a parametric query with a ‘?’ character as a placeholder for the argument. |
| **public** **void** doPrivilegedAction(    String username, **char**[] password  ) **throws** SQLException {    Connection connection = getConnection();  **if** (connection == **null**) {    }  **try** {      String pwd = hashPassword(password);  **if** (username.length() > 8) {        }        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");      } |

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

| **Principles(s):** The principle **Sanitize Data Sent to Other Systems** works here. This principle is valid as one would always want to sanitize variables and parameters before passing data into a SQL database to avoid a SQL injection. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| [Parasoft Jtest](https://wiki.sei.cmu.edu/confluence/display/java/Parasoft) | 2022.1 | **CERT.IDS00.TDSQL** | Protect against SQL injection |
| [SonarQube](https://wiki.sei.cmu.edu/confluence/display/java/SonarQube) | 6.7 | [S2077](https://rules.sonarsource.com/java/RSPEC-2077)  [S3649](https://rules.sonarsource.com/java/RSPEC-3649) | [Executing SQL queries is security-sensitive](https://rules.sonarsource.com/java/RSPEC-2077)  [SQL queries should not be vulnerable to injection attacks](https://rules.sonarsource.com/java/RSPEC-3649) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | STD-004-CPP | When memory is freed, all pointers become invalid. This can make the data at the freed location appear to be valid but change unexpectedly. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, ‘s’ is dereferenced after it has been deallocated. Usually, dynamic memory allocations and deallocations are far removed making it difficult to diagnose and recognize vulnerabilities. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In this compliant example, the dynamically allocated memory is not deallocated until it is no longer being used. |
| #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):** The principle **Architect and Design for Security** **Policies** applies here. This applies because when dealing with access, one may have to delete variables and arguments that may hold user data. While doing this, it is important to not access any deallocated memory until after it is no longer being used. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **483 S, 484 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Incorporate diagnostic tests using assertions** |
| --- | --- | --- |
| **Assertions** | STD-005-CPP | Assertions can be used to test for error conditions where any error codes should have been previously taken care of. |

| **Noncompliant Code** |
| --- |
| In this example of noncompliant code, the code contains an assert macro in an attempt to verify appropriate memory allocation. Doing this could lead to an abrupt termination of the process. |
| **char** \*dupstring(**const** **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char** \*)**malloc**(len + 1);  **assert**(NULL != dup);    **memcpy**(dup, c\_str, len + 1);  **return** dup;  } |

| **Compliant Code** |
| --- |
| In this example of compliant code, the code displays the correct way to check for memory exhaustion and appropriate memory allocation. |
| **char** \*dupstring(**const** **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char**\*)**malloc**(len + 1);    /\* Detect and handle memory allocation error \*/  **if** (NULL == dup) {  **return** NULL;    }    **memcpy**(dup, c\_str, len + 1);  **return** dup;  } |

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

| **Principles(s):** The principle **Use Effective Quality Assurance Techniques** fits best here. The quality assurance technique being used is testing the functionality of the code in the form of assertions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **LANG.FUNCS.ASSERTS** | Not enough assertions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ASSERT\_SIDE\_EFFECT** | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | **CERT\_C-MSC11-a** | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Do not abruptly terminate the program** |
| --- | --- | --- |
| **Exceptions** | STD-006-CPP | Abrupt and abnormal process termination can often lead to denial-of-service attacks. Do not implicitly or explicitly call the following terminate handlers: std::quick\_exit(),  std::abort(), or std::\_Exit() |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the call to function () was registered with an exit handler. This may result in a call to terminate because throwing function () may throw an exception. |
| #include <cstdlib>    **void** throwing\_function() noexcept(**false**);    **void** function() { // Not invoked by the program except as an exit handler.    throwing\_function();  }    **int** main() {  **if** (0 != std::**atexit**(function)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| In this complaint example, function () handles all exceptions thrown by throwing function () and does not rethrow. |
| #include <cstdlib>    **void** throwing\_function() noexcept(**false**);    **void** function() { // Not invoked by the program except as an exit handler.  **try** {      throwing\_function();    } **catch** (...) {      // Handle error    }  }    **int** main() {  **if** (0 != std::**atexit**(function)) {      // Handle error    }    // ...  } |

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

| **Principles(s):** The principle **Adopt a Secure Coding** Standard works bets here. When using the IDE or coding for the end of a program, it is important to adopt the practice of not abruptly terminating the program to prevent denial-of-service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: ERR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr50cpp.html) | Checks for implicit call to terminate() function (rule partially covered) |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **122 S** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-ERR50-a** **CERT\_CPP-ERR50-b** **CERT\_CPP-ERR50-c** **CERT\_CPP-ERR50-d** **CERT\_CPP-ERR50-e** **CERT\_CPP-ERR50-f** **CERT\_CPP-ERR50-g** **CERT\_CPP-ERR50-h** **CERT\_CPP-ERR50-i** **CERT\_CPP-ERR50-j** **CERT\_CPP-ERR50-k** **CERT\_CPP-ERR50-l** **CERT\_CPP-ERR50-m** **CERT\_CPP-ERR50-n** | The execution of a function registered with 'std::atexit()' or 'std::at\_quick\_exit()' should not exit via an exception Never allow an exception to be thrown from a destructor, deallocation, and swap Do not throw from within destructor There should be at least one exception handler to catch all otherwise unhandled exceptions An empty throw (throw;) shall only be used in the compound-statement of a catch handler Exceptions shall be raised only after start-up and before termination of the program 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 Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) Function called in global or namespace scope shall not throw unhandled exceptions Always catch exceptions Properly define exit handlers The 'abort()' function from the 'stdlib.h' or 'cstdlib' library shall not be used Avoid throwing exceptions from functions that are declared not to throw The 'quick\_exit()' and '\_Exit()' functions from the 'stdlib.h' or 'cstdlib' library shall not be used |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **BADFUNC.ABORT** **BADFUNC.EXIT** | Use of abort Use of exit |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Unsigned Integer Wrapping** |
| --- | --- | --- |
| Data Types | STD-007-CPP | Ensure that unsigned integers don’t wrap. Wrapping of unsigned integers are equivalent to regular unsigned integers and should be detected and prevented. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the addition of the unsigned integers could lead to an unsigned integer wrap. This could lead to an exploitable vulnerability due to the possible allocation of insufficient memory. |
| **void** func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum = ui\_a + ui\_b;  } |

| **Compliant Code** |
| --- |
| In this compliant example, a precondition test is performed to guarantee that there is no possibility of an unsigned wrap. |
| #include <limits.h>    **void** func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum;  **if** (UINT\_MAX - ui\_a < ui\_b) {      /\* Handle error \*/    } **else** {      usum = ui\_a + ui\_b;    }  } |

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

| **Principles(s):** The principle **Validate Input Data** would apply here. Although the data is not being entered by a user or by the system, unsigned integers could cause an exploitable vulnerability due to an issue with memory allocation. This would be similar to not verifying data values. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.1 | **CERT\_C-INT30-a** **CERT\_C-INT30-b** **CERT\_C-INT30-c** | Avoid integer overflows Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C: Rule INT30-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint30c.html) | Checks for:   * Unsigned integer overflow * Unsigned integer constant overflow   Rule partially covered. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **ALLOC.SIZE.ADDOFLOW** **ALLOC.SIZE.IOFLOW** **ALLOC.SIZE.MULOFLOW** **ALLOC.SIZE.SUBUFLOW** **MISC.MEM.SIZE.ADDOFLOW** **MISC.MEM.SIZE.BAD** **MISC.MEM.SIZE.MULOFLOW** **MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule by ensuring that operations are checked for overflow before being performed (Be mindful of exception INT30-EX2 because it excuses many operations from requiring [validation](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-validation), including all the operations that would validate a potentially dangerous operation. For instance, adding two unsigned ints together requires validation involving subtracting one of the numbers from UINT\_MAX, which itself requires no validation because it cannot wrap.) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Declaring a Reserved Identifier** |
| --- | --- | --- |
| Data Types | STD-008-CPP | Declaring an identifier that is reserved will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, because the library header includes <cstdint>, MAX\_SIZE conflicts with the name of the <cstdint> header used to denote the upper limit of size\_t. |
| #include <cinttypes> // for int\_fast16\_t    **void** function(std::int\_fast16\_t val) {  **enum** { MAX\_SIZE = 80 };    // ...  } |

| **Compliant Code** |
| --- |
| In this compliant example, the code doesn’t redefine the reserved name by removing MAX\_SIZE. |
| #include <cinttypes> // for std::int\_fast16\_t    **void** function(std::int\_fast16\_t val) {  **enum** { BufferSize = 80 };    // ...  } |

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

| **Principles(s):** The principle **Validate Input Data** would apply here. Although the data is not being entered by a user or by the system, declaring a reserved identifier would result in undefined behavior that could expose a vulnerability within the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreserved-id-macro -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.1p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL51-a** **CERT\_CPP-DCL51-b** **CERT\_CPP-DCL51-c** **CERT\_CPP-DCL51-d** **CERT\_CPP-DCL51-e** **CERT\_CPP-DCL51-f** | Do not #define or #undef identifiers with names which start with underscore Do not redefine reserved words Do not #define nor #undef identifier 'defined' The names of standard library macros, objects and functions shall not be reused The names of standard library macros, objects and functions shall not be reused (C90) The names of standard library macros, objects and functions shall not be reused (C99) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl51cpp.html) | Checks for redefinitions of reserved identifiers (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Side effects in unevaluated operands** |
| --- | --- | --- |
| Expressions | STD-009-CPP | Do not rely on side effects in unevaluated operands. Relying on the side effects of an operand that is not evaluated will result in unexpected behavior. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the expression ‘a++’ that is used to increment ‘a’ is not evaluated. |
| #include <iostream>  **void** function() {  **int** a = 14;  **int** b = **sizeof**(a++);    std::cout << a << ", " << b << std::endl;  } |

| **Compliant Code** |
| --- |
| In this complaint example, ‘a’ is incremented outside of the size\_of operator. |
| #include <iostream>  **void** function() {  **2**  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.1p0 | **LANG.STRUCT.SE.SIZEOF** | Side Effects in sizeof |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-EXP52-a** **CERT\_CPP-EXP52-b** **CERT\_CPP-EXP52-c** **CERT\_CPP-EXP52-d** **CERT\_CPP-EXP52-e** | The operand of the sizeof operator shall not contain any expression which has side effects Object designated by a volatile lvalue should not be accessed in the operand of the sizeof operator The function call that causes the side effect shall not be the operand of the sizeof operator The operand of the 'typeid' operator shall not contain any expression that has side effects The operand of the 'typeid' operator shall not contain a function call that causes side effects |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: EXP52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp52cpp.html) | Checks for logical operator operand with side effects |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **54 S, 133 S** | Partially implemented |

### 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 optimal tools to be used to cover each standard would be CodeSonar and Parasoft C/C++test. Almost all our standards are related to vulnerabilities that can be exploited via incorrect coding practices, it would be optimal to automate standard enforcement in the pre-production phase. These tools would be used during the build and verification and testing portion of the DevSecOps cycle. After the build, if any of the standards above are violated, the automation tools will throw an error to let the developer know that this isn’t allowed. These tools can be integrated into the IDE or CI/CD pipeline that is being used by the developers in the pre-production phase.

### 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 | Low | P3 | L3 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CPP | Low | Probable | Medium | P4 | L4 |
| STD-001-JAV | High | Probable | Medium | P12 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | Low | Unlikely | High | P1 | L3 |
| STD-006-CPP | Low | Probable | Medium | P4 | L3 |
| STD-007-CPP | High | Unlikely | High | P9 | L2 |
| STD-008 -PP | Low | Unlikely | Low | P3 | L3 |
| STD-009-CPP | Low | Unlikely | Low | P3 | 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 in rest applies to the practice of encrypting any data that is stored on a disk or a server. This would be used if you have any data that is backed up onto a system or if you are storing a hard copy of data anywhere. It would need to be encrypted so that if the copy of data does get into the wrong hands, the data wouldn’t be accessed. |
| Encryption at flight | Encryption in flight refers to data that is to be used on the network. So, this data is encrypted when you send it to a data point and is decrypted at the data point. The opposite transaction is then happening as you are receiving data back from the data point. This should be used when transferring data to and from data points so that if the data is intercepted at any point, the hacker will receive encrypted data that cannot be accessed. |
| Encryption in use | Encryption in use refers to the data that is being stored on a digital device. This isn’t the hard data but is the data that you have stored on your device before it is transmitted to another data point. This is necessary to protect from device hacking. If your digital device is hacked, all that is obtained would be encrypted data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication involves verifying user logins and credentials for anyone who is trying to access the system. This involves verifying login information, titles, and the version of the system that is being provided. |
| Authorization | Authorization refers to the access that each user should have. Each user shouldn’t need the same level of access based on their title so the system will need to verify that the data access varies between each user. |
| Accounting | Accounting is the process of keeping track of the data that is accessed by each user. This keeps track of the data that has been accessed, the amount of data being accessed, and any changes made to the system by the user. This information is usually kept in the form of an audit log which is helpful to have in the case of a hacking attempt or an attack. With accounting, it is much easier to see who was compromised, where the attack originated, and the dat that was accessed. |

**\***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 | 10/09/2022 | Project 1 | Alexis Alexander | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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