**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 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 | Validate input from all untrusted data sources. Proper input validation can eliminate vast majority of vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environment variables, and user controlled files. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. For example, if you system requires different privileges at different times, consider dividing the system into distinct intercommunicating systems. Each with their own distinguished privilege set. |
| 1. Keep It Simple | Keep the design as simple and small as possible. Complex designs increase the likelihood of errors and that they will be made in the implementation, configuration and use. Additionally the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex systems such as command shells, relational databases, and commercial off-the-shelf components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex system being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the system. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from being an exploitable vulnerability and limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions. |
| 1. Adopt a Secure Coding Standard | Develop or apply a secure coding standard for your target development language or platform. This means complying to all standards defined and sticking to them during the development process. Ensuring that a singular coding standard is followed helps ensure the application is secure under the standard and better yet provides consistency. |

### 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 | Ensure that when defining a variable (unsigned or signed), understand the bounds that will be required and the possibilities of those values going into the negatives or staying in the positive bounds. Using the smallest data type necessary is a good practice in order to avoid unaccounted for values or values that are misused with the data type. |

| **Noncompliant Code** |
| --- |
| This code fails to consider the if the value exceeds the specific data max range, a wraparound will occur. |
| 1 size\_t MAX\_RANGE = “value”  2 char example[MAX\_RANGE] = “values”  3 unsigned int sum = “value”  4 for (unsigned int i = 0; sum + i > MAX\_RANGE; i++) {  5 example[i] = sum;  6 sum += i;  7 } |

| **Compliant Code** |
| --- |
| This code will succeed and prevent any wraparound. Consider the use of similar data types with one another and also ensure that you apply variable types when necessary for specific purposes. If a value should and will never exceed a range, ensure that by limiting the value with the proper data type and also testing it(forcing the range to comply with logic). |
| 1 size\_t MAX\_RANGE = “value”  2 char example[MAX\_RANGE] = “values”  3 unsigned int sum = “value”  4 for (size\_t i = 0; i > MAX\_RANGE - sum; i++) {  5 example[i] = sum;  6 sum += i;  7 } |

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

| **Principles(s):** 1) Validate all User input in order to prevent data type overflow or corruption of data. 2) Use Testing techniques to ensure that the code itself is correctly assigned the proper variable type 3) Only use the specific type of data that is necessary for the real world use of the data (int for integer based items) |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | INTEGER\_OVERFLOW | Implemented |
| CodeSonar | 6.1p0 | 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 |
| Parasoft C/C++test | 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 | R2022b | CERT C: Rule INT30-C | Checks for  Unsigned integer overflow  Unsigned integer constant overflow |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Data values must adhere to proper ranging and allocation in order to prevent improper conversions and overflow. Their values must accommodate with to the size of the values being referenced. This is important as to not exceed the MAX or MIN values since this can cause wraparound. The rule INT31-C, ensure that integer conversions do not result in lost or misinterpreted data states that both implicit and explicit casting must be guaranteed not to result in lost or misinterpreted data. |

| **Noncompliant Code** |
| --- |
| Type range errors include loss of data and loss of sign, can occur when converting from a value of an unsigned integer type to a value of a signed integer type. The noncompliant code example below results in a truncation error due to overflow. |
| 1 unsigned long int example = ULONG\_MAX;  2 signed char example2;  3 example2 = (signed char)example; |

| **Compliant Code** |
| --- |
| It’s important to validate the ranges when converting from an unsigned type to a signed type. This compliant solution allows the conversion of unsigned long int value to value of signed char. |
| 1 unsigned long int example = ULONG\_MAX;  2 signed char example2;  3 if (example <= SCHAR\_MAX) {  4 example2 = (signed char)example;  5 }  6 else {  7 /\* handle error messages \*/  8} |

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

| **Principles(s):** 1) Validate user input ensuring that a max range or min range is possible to avoid overflow or access to inaccessible data. 2) Unit testing can avoid this altogether by throwing values out of the bounds to ensure that this is properly tested and quality assured. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Overflow\_upon\_dereference | Implemented |
| CodeSonar | 7.1p0 | 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 |
| Helix QAC | 2022.3 | C++2891, C++3139, C++3140 | Implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP\_CTR50-a | Guarantee that container indices are within the valid range |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Copying data to a buffer that is not large enough to hold data results in buffer overflow. Buffer overflows occur when manipulating a string and the range of the size of the array is not accounted for. In order to prevent the error, it’s important to limit the size of the array using truncation or ensure that the destination is of sufficient size to hold the data copied, including the null terminator. |

| **Noncompliant Code** |
| --- |
| This noncompliant example demonstrates an off-by-one error. The loop copies data from src to dest, but because the loop does not account for the null-termination character, it may be incorrectly written 1 byte past the end of dest. |
| 1 size\_t i;  2 for (i = 0; src[i] && (i < n); ++i) {  3 dest[i] = src[i];  4 }  5 dest[i] = ‘\0’; |

| **Compliant Code** |
| --- |
| In this compliant example, the loop termination condition is modified to account for the null-termination character that is append to dest |
| 1 size\_t i;  2 for (i = 0; src[i] && (i < n - 1); ++i) {  3 dest[i] = src[i];  4 }  5 dest[i] = ‘\0’; |

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

| **Principles(s):** 1) Make sure to range check in order to ensure that the range does not exceed the maximum 2) Unit test for these values and ensure that the values are contained in a size that fits the data types |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Assert\_failure | Implemented |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Helix QAC | 2022.3 | C++3162, C++3163, C++3164, C++3165 | Implemented |
| Parasoft | 2022.1 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | SQL Injection involves the use of unfiltered and partialized query string values which can be included and expanded upon during execution of the query. This includes adding query keywords and variables into the query itself, which can force a query to be true and therefore bypass the security of the system or access normally inaccessible data. String concatenation of queries is the primary point for sql injection scripting attacks, since there is no use of actual dynamic query creation and simply injecting the query values. A solution to this issue involves using regex variable to parse out any values or query language which does not comply with the query standards (this means accepting only regex that complies with the query standard). |

| **Noncompliant Code** |
| --- |
| This example showcases an sql injection occurring without the checks involved for verifying query concatenation values. This statement simply executes the query, and if successful returns true, otherwise throws an error. This does not comply with standards and does not check for unnecessary query parameters which could exploit the query. |
| 1 records.clear()  2 if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_messages) != SQLITE\_OK){  3 sqlite3\_free(error\_message);  4 return false;  5 }  6 return true; |

| **Compliant Code** |
| --- |
| This example showcases mitigation of the sql injection attempt through the use of regex expressions, which involves verifying that the query is formatted specifically to a standard specified using the regex expression. If the expression does not comply and the query data is not followed, it will throw an error and notify the user of sql injection attempt. |
| 1 Std::regex RegexExpression("[^0-9A-Z+$]\\s\*(and|or)\\s\*[\\'\"]?", std::regex\_constants::icase);  2 if (std::regex\_search(sql, SQLInjectionCriteria)) {  3 std::cout << "SQL Injection detected: The following error is associated -> " << &error\_message << std::endl;  4 return false;  5 }  6 records.clear()  7 if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_messages) != SQLITE\_OK){  8 sqlite3\_free(error\_message);  9 return false;  5 }  6 return true; |

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

| **Principles(s):** 1) Validate user input for constructing SQL queries and parameterization. 2) unit test for the ability to breach the SQL data and ensure that no data can bypass checks, 3) Use layered defensive measures which include adding multiple layers to tackle different aspects of SQL injection |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.MEM.BO  LANG.MEM.TO  MISC.MEM.NTERM  BADFUNC.BO.\* | Buffer overrun  Type overrun  No space for null terminator  A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Implemented |
| Parasoft C/C++ | 2022.1 | CERT\_C-STR31-a  CERT\_C-STR31-b  CERT\_C-STR31-c  CERT\_C-STR31-d  CERT\_C-STR31-e | Avoid accessing arrays out of bounds  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Avoid using unsafe string functions which may cause buffer overflows |
|  |  |  |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Evaluating pointers which includes dereferencing them, and using it as an arithmetic operator while using it at the right hand side of assignment into memory which has already been deallocated can cause undefined behavior. This means the pointer is dangling and the memory has already previously been deallocated. According to C standards, using the value of a pointer that refers to space deallocated by the call of free() or realloc() is undefined behavior. The pointer value is indeterminate and is at the memory managers discretion. |

| **Noncompliant Code** |
| --- |
| An example of accessing freed memory involves freeing p before p->next occurs which removes the value of p, and p->next then reads memory which has already been freed. |
| 1 struct node {  2 int value;  3 struct node \*next;  4 };  5 for (struct node \*p = head; p != NULL; p->next) {  6 free(p);  7 } |

| **Compliant Code** |
| --- |
| It’s important to store the next node value prior to freeing the memory in order to avoid accessing freed memory. |
| [1 struct node {  2 int value;  3 struct node \*next;  4 };  5 struct node \*q;  6 for (struct node \*p = head; p != NULL; p->next) {  7 q = p->next;  8 free(p);  9 } |

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

| **Principles(s):** 1) Ensure that data that has been freed is not used in a part of the program, 2) Validate the data exists prior to using it to ensure that the data itself is valid and holds value 3) Unit test the values and the flow/behavior of the application to ensure that the data is not freed before being accessed |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Dangling\_pointer\_use | implemented |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 7.1p0 | ALLOC.UAF | Use after free |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assert statements are good for detecting and handling errors properly in a safe manner. Test for conditions which should be deemed false. Exceptions should not be used since the condition involves interrupting the flow of the program, and should only indicate code that needs to be fixed. It’s good practice to test and log these errors for previous issues found in order to avoid them coming up again in future code. Assert statements make a great use of ensuring that something is valid in the code. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a call to an exit was referenced prior to an assert statement, which could have reported the error. |
| 1 if (i != 1) {  2 exit()  3 } |

| **Compliant Code** |
| --- |
| The code example below complies with coding standards for assertions and enables the developers to gain insight as to when an error or variable is misbehaving. |
| 1 static\_assert((i != 1), “ERROR: assertion has been throw for value of I where i is not 1”);  2 /\* continue flow of application while raising assertion statement \*/ |

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

| **Principles(s):** 1) Ensure that assertions are properly used over exceptions or error handling 2) Unit test for if an assertion is thrown and validate the message thrown 3) Gracefully allow the user to end the program and provide context over abruptly closing the application due to a crash |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Stdlib-use | Partially checked |
| CodeSonar | 7.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Parasoft C/C++test | 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 |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | When an exception is thrown, control is transferred to the nearest handler with a type that matches the type of the exception thrown. If no matching handler is directly found within the handlers for a try block in which the exception is thrown, the search for a matching handler continues to dynamically search for handlers in the surrounding try blocks. All exceptions thrown by an application must be caught by a matching exception handler. Even if the exception cannot be gracefully recovered from, using the matching exception handler ensuring that the stack will be properly unwound. As per ERR50-CPP-EX1, a program that encounters an unrecoverable exception may explicitly catch the exception and terminate, but it may not allow the exception to remain uncaught. |

| **Noncompliant Code** |
| --- |
| The below code does not comply with exception handling. Neither main or the function itself handle exceptions thrown and therefore the terminate() method is called. |
| 1 void function() noexcept(false);  2 int main() {  3 function();  4 } |

| **Compliant Code** |
| --- |
| The below code handles all exceptions which ensures that the stack is unwound to the main and function which allows for a graceful management of external resources |
| 1 void function() noexcept(false);  2 int main() {  3 try {  4 function();  5 } catch (…) {  6 /\* Handle exceptions \*/  7 } |

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

| **Principles(s):** 1) Handle all exceptions with the appropriate handler and exception with a descriptive message 2) Never completely stop the execution of an application but ensure that the exceptions are captured 3) Unit test for proper exception handling and expected outcomes |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| CodeSonar | 7.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| LDRA tool suite | 9.7.1 | 527 S | Partially Implemented |
| Parasoft C/C++test | 2022.1 | 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 |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | STD-008-CPP | Do not delete an object of derived class type through a pointer to its base class type that has a non-virtual destructor. Instead, the base class should be defined with virtual destructor. Deleting an object through pointer to a type without a virtual destructor results in undefined behavior. The C++ standard states the following: If a class has no user-declared destructor, a destructor is implicitly declared as defaulted. An implicitly declared destructor is an inline public member of its class. |

| **Noncompliant Code** |
| --- |
| Example a below is a polymorphic pointer type whose static type is Base \* and whose dynamic type is derived \*. When a is deleted, it results in undefined behavior because Base does not have a virtual destructor. |
| 1 struct Base {  2 virtual void function();  3 };  4 struct Derived : Base {};  5 void function() {  6 Base a\* = new Derived();  7 delete a;  8 } |

| **Compliant Code** |
| --- |
| This compliant code below, the destructor for Base has an explicitly declared virtual destructor, ensuring that the polymorphic delete operation results in well-defined behavior. |
| 1 struct Base {  2 virtual ~Base() = default;  3 virtual void function();  4 };  5 struct Derived : Base {};  6 void function() {  7 Base a\* = new Derived();  8 delete a;  9 } |

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

| **Principles(s):** Always use destructors when creating a constructor for an associated virtual function. 2) Unit test to ensure that the destructor is actually handled and works properly with expected behavior 3) Ensure the object exists prior to calling a destructor |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | non-virtual-public-destructor-in-non-final-class | Partially Checked |
| CodeSonar | 7.1p0 | LANG.STRUCT.DNVD | Delete with Non-Virtual Destructor |
| LDRA tool suite | 9.7.1 | 303 S | Partially Implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-OOP52-a | Define a virtual destructor in classes used as base classes which have virtual functions |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | STD-009-CPP | Using and maintaining files can be a lengthy operation, and if the files are not properly closed, the file descriptors themselves are occupied and could potentially fail or crash your program. It can also leave vulnerabilities for leaving a file in the open state since this allows for any access to a file to be still accessible. Properly managing an object of types basic\_ifstream is sufficient with the use of open() and close(). |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, a fstream object file is constructed. The file is then remaining open, while the application is terminated, resulting in the file not being properly closed. |
| 1 std::fstream file(fileName);  2 if (!file.isopen()) {  3 // handle the error  4 return;  5 }  6 std::terminate();  } |

| **Compliant Code** |
| --- |
| In this example below, the close method is called prior to the terminate method is called, ensuring that the file is properly handled. This protects the data of the file as well as the runtime of the file descriptor. |
| 1 std::fstream file(fileName);  2 if (!file.isopen()) {  3 // handle the error  4 return;  5 }  6 file.close();  7 if (file.fail()) {  8 // handle the error  9 }  10 std::terminate();  } |

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

| **Principles(s):** 1) Ensure that files are closed after opened and handle proper errors on closure and opening 2) Unit test for the closure of the file and ensure that the file does not throw any errors when operating |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft Insure++ |  |  | Runtime detection |
| Polyspace Bug Finder | R2022b | CERT C++:FIO51-CPP | Checks for resource leak (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Pseudorandom Numbers Standard | STD-010-CPP | Pseudorandom number generators use mathematical algorithms to produce a sequence of numbers with good statistical properties. These numbers are not genuinely random though. The c standard rand() function exposed through the C++ library makes no guarantees as to the quality of the random sequence produced. The numbers generated by some implementations have comparatively short cycle, and the numbers can be predictable. |

| **Noncompliant Code** |
| --- |
| The following non compliant code generates an ID with a numeric part produced by calling the rand() function. The ID’s produced are predictable and have limited randomness. |
| 1 Std::string id(“ID”);  2 id += std::to\_string(std::rand()%10000); |

| **Compliant Code** |
| --- |
| The C++ library provides mechanisms for fine-grained control over pseudorandom number generation. Two parts: one is the algorithm responsible for providing the random values, and the other is responsible for distribution of the random values via a density function. This soluition uses the Mersenne Twister algorithm as the engine for generating random values and a uniform distribution to negate bias. |
| 1 Std::string id(“ID”);  2 Std::uniform\_int\_distribution<int> distribution(0, 10000);  3 Std::random\_device rd;  4 std::mt19937 engine(rd());  5 id += std::to\_string(distribution(engine)); |

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

| **Principles(s):** 1) Ensure Pseudorandom numbers are truly random through the use of a 3rd party random distribution 2) Ensure the security of the randomness by separating a distribution from the engine in order to generate a true random number 3) unit test that a distribution and engine are created (since not possible to test random number itself) |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Bad-function (AUTOSAR.26.5.1A) | Fully checked |
| Clang | 4.0 (prerelease) | Cert-msc50-cpp | Checked by clang-tidy |
| CodeSonar | 7.1p0 | BADFUNC.RANDOM.RAND | Use of rand |
| ÉCLAIR | 1.2 | CC2.MSC30 | Fully 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.

Automation can be created upon the Build stage by automating any manual processes into CICD pipelines, which will greatly reduce the manual work done by the team as well as increase workflow. Using Docker containers, GitLab, Azure Pipelines, and or Jenkins for CI.

Verifying and testing in SecOps can be done through automating virtualization of containers and deployment. Implementing automation and security tests along with regression testing will go a long way towards securing the CD aspect of development.

Monitoring and detection can be handled automatically through static application security testing during builds on specific sections to test for of the build. It’s important to incorporate real time testing in the SDLC process to look for real time vulnerabilities.

Utilizing tools such as OWASP Dependency-Check along with other tools will help with checking code for dependencies and vulnerabilities.

### 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 | High | 2 |
| STD-002-CPP | High | Likely | High | High | 2 |
| STD-003-CPP | High | Unlikely | Medium | Medium | 2 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Probable | Medium | Low | 3 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | Low | Likely | Low | Medium | 2 |
| STD-009-CPP | Medium | Unlikely | Medium | Low | 3 |
| STD-010-CPP | Medium | Unlikely | Medium | Low | 2 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### 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 used for protecting data that is at rest, using encrypting all data at rest using full-disk encryption as well as the data used at both the system level and server level along with the databases. |
| Encryption at flight | Encryption at flight is data that is protected during transfer or travel over a network. Using up to date and secure libraries along with encryption keys that have both a public and private key allows for end-to-end protection. Using Managed File Transfer or SSH with an expiration data will also keep data in flight protected along with the expiration being on the link and using data leak preventative measure that could potentially built into the cloud services provider itself. |
| Encryption in use | Utilizing identity management mechanics to confirm access and identity along with timed access will prevent users from becoming vulnerabilities themselves. IRM digital management can also apply protection. Encryption in use means that an active user is navigating, using, or consuming the data and should have the required rights and role to access such data, along with their session being encrypted or protected in some manner to avoid Man in the Middle Attacks. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of managing, measuring, and accepting users to access a system in a valid form that identifies the current user looking for access. The user must have valid credentials and provide a valid passphrase as well as instant authentication methods such as 2FA or MFA. This ensures that a user has the proper credentials as well as identifies that it is in fact the user and not a system or third party attempting to breach the system. |
| Authorization | Once a user has been authenticated the user must be authorized to access the data. This is determined on the rights and proper permissions allowed for the user. This is where the principle of least privilege comes into play. The user will either be granted minimal access or timed access. |
| Accounting | Monitoring and acknowledging each access or entry into the system through a systematic approach allows for the auditing of users and who has current access to which systems along with date, time, and logging information. This ensures that resources are monitored and create a trail for specific actions or interactions with data. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 12/03/2022 | Completed Entirety of the Document | Blaze Halderman | Professor Demory |
|  |  |  |  |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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