**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 | This principle involves the process of checking all data received via input before processing. Parts of input validation include checking string lengths, ensuring that the data is within an expected range (to prevent overflows), making sure that the input information is the correct type of data expected, etc. By properly performing these checks, vulnerabilities that could arise from buffer overflows, injection attacks, etc, are reduced greatly. This process does not just include raw input given from a user, but also inputs from sources such as other functions, network inputs, and really any area where an external data source can be introduced to the code. |
| 1. Heed Compiler Warnings | This principle is a reminder of the importance of monitoring the compiler as it outputs warnings. These warnings can be early indications of security vulnerabilities that could arise due to factors such as out-of-date dependencies and issues introduced due to issues in the code itself. Some examples of things the compiler can catch are uninitialized variables, type-conversion issues, and areas prone to buffer overflow. |
| 1. Architect and Design for Security Policies | This principle is an assertion that has been foot stopped time and time again. When developing an application, it is important to build it with security features from the beginning and to ensure security considerations are factored into the entire design process. It is important to perform such planning for security even at early stages to avoid leaving security as an afterthought, which then postures the code to have a reactive approach to security vulnerabilities rather than a proactive approach. |
| 1. Keep It Simple | Code is inherently complicated, even when presented at a baseline level. This principle suggests that the developer avoid using overly complicated code structure because that code can be difficult to maintain, review, and test for issues. This can be achieved by resorting to C++ standard library features whenever possible and avoiding convoluting the code for the sake of optimization. In the world of modern computing, these optimizations often have little overall net gain for the end user, so it is better to focus on code that is easier to maintain and thus easier to secure. |
| 1. Default Deny | This principle asserts that the developer should only grant accesses as needed and to present the most restrictions possible from the get go. The “default deny” mantra simply means that all accesses are denied to the end user by default and to design a system in such a way that the user only is given access to what they need to perform their tasks and nothing more. Similar to government security concepts such as “need to know”. If you don’t have the “need to know”, then you will not have access to that information. |
| 1. Adhere to the Principle of Least Privilege | Similar in concept to the default deny principle, this principle states that the end user should only be given the minimum privileges that they need to accomplish their operations. This can be achieved via careful management of access to sensitive resources, and ensuring that even at the component level there is no access to data and functions that are not necessary to the operation of the specific task. |
| 1. Sanitize Data Sent to Other Systems | This principle deals with the output information from the application. Once the task/operation has been accomplished, it is important to ensure that outputted data is both protected and clean of sensitive data. One place that sensitive information can be leaked is from information sent through error messaging or logging. It is critical to make sure that none of these weak points will cause issue and to also ensure that any information that needs protecting is properly encrypted before being transmitted to an external source. |
| 1. Practice Defense in Depth | This principle accepts that a sole security design feature or function is not enough to protect an entire system. The developer must ensure that they incorporate many layers of security across different levels of the system. A few of the prior principles are a single piece of the puzzle; incorporating multiple is the practice of defense in depth and ensures a secure system. For example, practicing good input validation as well as sanitizing data sent to other systems would be two separate security features that combined would only make the system more locked down and safe from data breach. |
| 1. Use Effective Quality Assurance Techniques | This principle marks the importance of testing practices when building out a system. Throughout the QA process, there should always be security testing on top of the standard tests for functionality. This sort of testing prevents vulnerabilities from being introduced into production in the first place and goes hand-in-hand with the principle for architecting and designing for security policy. |
| 1. Adopt a Secure Coding Standard | Using a standard framework for secure coding allows for the development team to speak the same language and operate within the same frameworks. This not speeds the development process because testing can be done more rapidly, but it also prevents security vulnerabilities from arising due to different implementations across the system to combat similar issues. The guidelines in a coding standard also prevent poor coding practices from unintentionally presenting security vulnerabilities into the system. |

### 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** |
| --- | --- | --- |
| **Do not depend on the order of evaluation for side effects** | [EXP-050-CPP] | Practicing sound expression management serves to prevent errors and crashing, which can lead to sensitive information being leaked from a system. There are a multitude of guidelines and rules for how to prevent issues due to expressions, such as by not reading uninitialized memory or not depending on the order of evaluation for side effects. EXP50-CPP through EXP63-CPP in the SEI CERT C++ Coding standard all apply to expressions. EXP50 specifically highlights issues that can arise from not specifically ensuring that side-effect actions (which are actions that modify the state of the execution environment). This essentially means that the developer must be aware of what the defined behavior of certain expressions will do so that unintended vulnerabilities are not introduced. |

| **Noncompliant Code** |
| --- |
| `i` is evaluated more than once in an unsequenced manner , so the behavior of the expression is undefined. |
| Void f(int i, const int \*b) {  Int a= i+b[++i];  // ..  } |

| **Compliant Code** |
| --- |
| This example is independent of the order of evaluation of the operands and can only be interpreted in one way. |
| Void f(int i, const int \*b) {  Int a=i + b[i+1];  ++i;  // ..  } |

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

| **Principles(s):** Sanitize data sent to other systems, practice defense in depth, validate input data: by ensuring that expressions have defined and no unintentional behavior, there are less opportunities for unsecure or incorrect data being sent to other systems. This level of protection is also another facet of practicing defense in depth, as it alone will not secure a system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.SE.DEC**  **LANG.STRUCT.SE.INC** | Side effects in expression with decrement  Side effects in expression with increment |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | V7.5.0 | **EVALUATION\_ORDER** | Can detect the specific instance where a statement contains multiple side effects on the same value with an undefined evaluation order because, with different compiler flags or different compilers or platforms, the statement may behave differently |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CCS.EXP30** | Fully implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Do not cast to an out-of-range enumeration value** | [INT-050-CPP] | This standard refers to the issues that can arise from operating on either unspecified values or on a value that is out-of-range of the enumeration type. C++ enumerations can either be scoped (where type is fixed) or unscoped (where type may or may not be fixed). Arithmetic values must be cast to be within the range of values that the enumeration can represent. |

| **Noncompliant Code** |
| --- |
| Checks whether a given value is within range of acceptable enumeration values, however, it does so after casting to the enumeration type, which may not be able to represent the given integer value. |
| Enum EnumType {  First,  Second,  Third  };  Void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  If(enumVar < First || enumVar > Third) {  // Handle Error  }  } |

| **Compliant Code** |
| --- |
| Checks that the value can be represented by enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value. It does so by restricting the converted value to one for which there is a specific enumerator value. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** validate input data: Ensuring that unspecified or out-of-range values are not used prevents crashing and the creation of unexpected behavior, which could then be exploited and potentially used to extract further information. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **Cast-integer-to-enum** | Partially Checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters value  Cast alters value |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-INT50-a** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **Cast-integer-to-enum** | Partially Checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Do not invoke virtual functions from constructors or destructors** | [OOP-050-C++] | Object-oriented programming can be of big benefit to developers because it can help to manage and simplify processes that require the creation of multiple data types that contain large quantities of information. OOP can also introduce a lot of vulnerabilities though, as the process of implementing good OOP can become a complex process in its own right. This standard refers to issues that can arise from the invocation of a virtual function from the baseline building blocks of OOP (constructors, which build new objects, and destructors, which tear down and clean up). Because of the process C++ takes when constructing objects (where it starts with base classes and moves to derived), trying to call a derived class can be dangerous because it has not had its resources initialized yet. |

| **Noncompliant Code** |
| --- |
| The base class attempts to seize and release an object’s resource through calls to virtual functions from the constructor and destructor. The B:B() constructor calls B::seize() instead of D::seize(). B::~B() calls B::release() instead of D::release(). No derived class resources will be seized or released during the initialization and destruction of the objects in this code. This would lead to undefined behavior. |
| struct B {    B() { seize(); }    virtual ~B() { release(); }    protected:    virtual void seize();    virtual void release();  };    struct D : B {    virtual ~D() = default;    protected:    void seize() override {      B::seize();      // Get derived resources...    }      void release() override {      // Release derived resources...      B::release();    }  }; |

| **Compliant Code** |
| --- |
| The constructors and destructors call a nonvirtual, private member function instead of calling a virtual function. Each class is responsible for seizing and releasing its own resources |
| class B {    void seize\_mine();    void release\_mine();    public:    B() { seize\_mine(); }    virtual ~B() { release\_mine(); }    protected:    virtual void seize() { seize\_mine(); }    virtual void release() { release\_mine(); }  };    class D : public B {    void seize\_mine();    void release\_mine();    public:    D() { seize\_mine(); }    virtual ~D() { release\_mine(); }    protected:    void seize() override {      B::seize();      seize\_mine();    }      void release() override {      release\_mine();      B::release();    }  }; |

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

| **Principles(s):** Keep it simple, use effective quality assurance techniques: This code standard can be prevented by performing effective setup and cleanup of OOP concepts, which is something that should fall into the category of keeping things simple. By introducing more complex systems such as OOP into a system, it is imperative that quality assurance techniques are implemented to check for easy mistakes such as this. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **Virtual-call-in-constructor** | Fully Checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | Cland-analyzer-alpha.cplusplus.VirtualCall | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.VCAL\_IN\_CTOR**  **LANG.STRUCT.VCALL\_IN\_DTOR** | Virtual call in constructor  Virtual call in destructor |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **476 S, 92 D** | Fully Implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection**  **Sanitize data passed to complex subsystems** | [STR-002-C] | SQL injection are vulnerabilities that can appear when elements of a SQL query originate from an untrusted source. This untrusted data could cause alterations to a query and result in data leaks or modification. A common place that this may occur is in username/password input.  This issue is part of a wider C standard about the sanitization of data that is passed to complex subsystems. Any string data that is passed to such a subsystem could cause issues. |

| **Noncompliant Code** |
| --- |
| Sanitization of data first means that the developer must understand the data being passed and its capabilities. This example is for an application that inputs an email address to a buffer and uses the string as an argument to call to `system()` |
| Sprint(buffer, “/bin/mail %s < /tmp/email”, addr);  System(buffer); |

| **Compliant Code** |
| --- |
| Ensure that all valid data is accepted while potentially dangerous data is either rejected or sanitized. Creating a whitelist of acceptable characters and removing unacceptable characters is a common method for solving this issue |
| Static char ok\_chars[] = “abcdefghijklmnopqrstuvwxyz”  “ABCDEFGHIJKLMNOPQRSTUVWXYZ”  “1234567890\_-.@”;  Char user\_data[] = “Bad char 1:} Bad char 2:{“;  Char \*cp = user\_data; /\* Cursor into string \*/  Const char \*end = user\_data + strlen(user\_data);  For (cp += strspn(cp, ok\_chars); cp != end; cp+= strspn(cp, ok\_chars)) {  \*cp = ‘\_’;  } |

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

| **Principles(s):** Validate input data, sanitize data sent to other systems, practice defense in depth: Ensure all user input data is validated, especially when sending to a complex system like SQL. The sanitization of such inputs also prevents the potential for SQL injection attacks. This is one of many layers of protection on a system which loops back to the concepts of practicing defense in depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | |  |  | | --- | --- | |  | **IO.INJ.COMMAND IO.INJ.FMT IO.INJ.LDAP IO.INJ.LIB IO.INJ.SQL IO.UT.LIB IO.UT.PROC** | | Command injection  Format string injection  LDAP injection  Library injection  SQL injection  Untrusted Library Load  Untrusted process creation |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-STR02-a**  **CERT\_C-STR02-b**  **CERT\_C-STR03-c** | Protect against command injection  Protect against file name injection  Protect against SQL injection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | CERT-C: Rec.STR02-C | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path   Rec. Partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection**  **Detect and handle memory allocation errors** | [MEM-052-CPP] | It is necessary to protect and monitor memory allocation and management throughout a program’s life cycle so that undefined behaviors do not present. MEM-052-CPP in the SEI Coding standard refers specifically to the detection and handling of memory allocation errors, which if not taken care of, can either lead to undefined behavior or crashing. |

| **Noncompliant Code** |
| --- |
| An array of int is created and the results of the allocation are not checked. An exception could be thrown and a crash of the program could occur. |
| #include <cstring>  Void f(const int \*array, std::size\_t size) noexcept {  Int \*copy = new int[size];  Std::memcpy(copy, array, size \* sizeof(\*copy));  // …  Delete [] copy;  } |

| **Compliant Code** |
| --- |
| This solution handles error conditions appropriately if the returned pointer is nullptr. It is important to always test to ensure a returned pointer is not a nullptr before referencing it. |
| #include <cstring>  #include <new>  Void f(const int \*array, std::size\_t size) noexcept {  Int \*copy = new (std::nothrow) int[size];  If (!copy) {  //Handle error  Return;  }  Std::memcpy(copy, array, size \* sizeof(\*copy));  // …  Delete [] copy;  } |

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

| **Principles(s):** validate input data, keep it simple, practice defense in depth: any input data, including memory allocation devices such as pointers, need to be validated to ensure no unexpected behavior or issues. C++ features like smart pointers can help with concepts of keeping things simple and reduce memory management complexity. This is another coding standard that would contribute to defense in depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Coverity) | 7.5 | **CHECKED\_RETURN** | Finds inconsistencies in how function call return values are handled |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP\_MEM52-a**  **CERT\_CPP\_MEM52-b** | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function’s parameters is undefined |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | **CERT C++: MEM52-CPP** | Checks for unprotected dynamic memory allocation (rule partially covered) |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions**  **Use a static assertion to test the value of a constant expression** | [DCL-003-C] | Assertions are critical to debugging and are an excellent validation tool. They ensure a program is executing correctly and can catch logic errors.  They should not be used for normal error handling but are a great tool during the development and testing phases of a program’s life cycle.  DCL-003-C concerns the use of static assertions in C. |

| **Noncompliant Code** |
| --- |
| The code here uses assert() to check a property concerning memory-mapped structures that are essential. In this example, the macro is not placed in the proper location; it is far away from the definition of the structure it is referring to and only will occur at runtime if the code path containing the assertion is executed. |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)); |

| **Compliant Code** |
| --- |
| This example has a static assertion that is not hidden in a function, thus ensuring its execution. This means issues can be assessed at compile time rather than runtime. |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int), “Structure most not have any padding”); |

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

| **Principles(s):** Heed compiler warnings, Use effective quality assurance techniques: Properly placed assertions will provide insights during the compilation process. Assertions are also a standard validation and testing technique across multiple platforms and programming languages. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | Misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions**  **Handle all exceptions** | [ERR-051-CPP] | It is important to handle all exceptions in a secure and forthright manner so that there is not potential information leakage. Further, if an exception handler is not found that matches the type of exception, then C++ will call std::terminate() by default, which would terminate the program (a crash). Abnormal process termination such as this is a typical route for denial-of-service attacks. |

| **Noncompliant Code** |
| --- |
| In this code, neither f() nor main() will catch the exceptions that are thrown by throwing\_func(). Because of this, the program will abnormally terminate. |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  f();  } |

| **Compliant Code** |
| --- |
| This revision handles the exceptions when they occur, which allows for the correct management of external resources and no abnormal termination of the program. |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  try {  f();  } catch (…){  // handle error  }  } |

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

| **Principles(s):** Validate input data, sanitize data sent to other systems, practice defense in depth: This coding standard ensures that exception data (including input data) is not exposed. This also keeps the data sanitized before being logged or displayed. This security measure is one of many to ensure defense in depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **Main-function-catch-all**  **Early-catch-all** | Partially Checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | **CERT C++: ERR51-CPP** | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **Main-function-catch-all**  **Early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Do not alternately input and output from a file stream without an intervening positioning call | [FIO-050-CPP] | Not properly handling input and output from a file stream can result in undefined behavior. This is due to C++/C standards. C++ standards mention that restrictions on reading and writing a sequence by an object of class basic\_filebuf<charT, traits> are the same as for reading and writing in the standard C library, which specifies that output cannot be directly followed by input without an intervening call to fflush or a file positioning function. |

| **Noncompliant Code** |
| --- |
| This code appends data to the end of a file and then reads from the same file. Since there is no positioning call between the formatted output and input calls, the behavior is undefined |
| #include <fstream>  #include <string>  Void f(const std::string &filename) {  std::fstream file(filename);  if(!file.is\_open()) {  //Handle error  return;  }    File << “Output some data”;  std::string str;  file >> str;  } |

| **Compliant Code** |
| --- |
| To fix, the std::basic\_istream<T>::seekg() function is called between the output and input, which eliminates the undefined behavior. |
| #include <fstream>  #include <string>  void f(const std::string &filename) {  std::fstream file(filename);  if (!file.is\_open()) {  // handle error  return;  }  file << “Output some data”;  std::string str;  file.seekg(0, std::ios::beg);  file >> str;  } |

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

| **Principles(s):** validate input data, architect and design for security policies, sanitize data sent to other systems: this level of validation ensures that input data going to a file is not undefined. This is good architecting and design for security policies because it addresses C++/C standard practices. Finally, data being written to a file typically suggests that it will go to another system at some point, and this ensures unexpected information and behavior is not sent to such systems. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **IO.IOWOP**  **IO.OIWOP** | Input after output without positioning  Output after input without positioning |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-FIO50-a** | Do not alternately input and output from a stream without an intervening flush or positioning call |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | **CERT C++:FIO50-CPP** | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-FIO50** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Ensure actively held locks are released on exceptional conditions | [CON-051-CPP] | Mutexes can be used to protect accesses to shared data and may be locked using the lock() function and unlocked with unlock(). If an exception occurs between those calls, and this changes the flow so that no unlock() is called, the mutex will be left in a locked state. No critical sections protected by that mutex will thus be allowed to execute. This likely will lead to a deadlock. Because of this, it is important to ensure that no throwing of an exception will cause a mutex to remain locked indefinitely. |

| **Noncompliant Code** |
| --- |
| This code manipulates shared data and protects the critical section by locking the mutex. Normally, it would unlock the mutex once execution is finished, however, if an exception occurs, the mutex will remain locked. |
| #include<mutex>  void manipulate\_shared\_data(std::mutex &pm) {  pm.lock();  //Perform work on shared data.  Pm.unlock();  } |

| **Compliant Code** |
| --- |
| This edit modifies the previous code such that any exceptions that may be thrown when working on the shared data will unlock the mutex before rethrowing the exception |
| #include <mutex>  void manipulate\_shared\_data(std::mutex &pm) {  pm.lock();  try {  //Perform work on shared data  } catch (…) {  pm.unlock();  throw;  }  pm.unlock(); //in case no exceptions occur  } |

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

| **Principles(s):** Adhere to the principle of least privilege, keep it simple, validate input data: keeping data locked when not necessary would deny access to data as required, thus not adhering to principle of least privilege; this is also a input data concern, as passing the shared data but locked would be unexpected behavior and could cause system issues. This also highlights the importance of adhering to best practices in order to keep things simple- this prevents confusion by unintentionally locking data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **CONCURRENCY.LOCK.NOUNLOCK** | Missing Lock Release |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.4 | **C++5018** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-CON51-a** | Do not call lock() directly on a mutex |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | **CERT C++:CON51-CPP** | Checks for lock possibly not released on exception (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Do not use std::rand() for generating pseudorandom numbers | [MSC-050-CPP] | While a pseudorandom number generator can produce a sequence of numbers with good statistical properties, the numbers are not genuinely random. C standard library rand() does not guarantee the quality of the random sequence; they can be predictable. This means that these values can be exploited or figured out. |

| **Noncompliant Code** |
| --- |
| This code generates an ID with a numeric part that is created with rand(). The IDs produced are thus predictable and have a limited randomness to it. |
| #include <cstdlib>  #include <string>  void f() {  std::string id(“ID”); // holds the ID, starting with the characters “ID” followed by a random integer in the range [0-10000]  id += std::to\_string(std::rand() % 10000);  // …  } |

| **Compliant Code** |
| --- |
| C++ standard library provides mechanisms for breaking random number generation into first an algorithm to provide the values (the engine) and second the distribution of such random values (the distribution). This ensures that values are properly distributed within a given range and bias issues are no longer a concern. |
| #include <random>  #include <string>  Void f() {  std::string id(“ID”); //holds the ID, starting with the characters “ID” followed by a random integer in the range [0-10000]  std::uniform\_int\_distribution<int> distribution(0, 10000);  std::random\_device rd;  std::mt19937 engine(rd());  id += std::to\_string(distribution(engine));  // …  } |

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

| **Principles(s):** Validate input data, sanitize data sent to other systems, practice defense in depth: This prevents unexpected behavior by ironically ensuring that the behavior is unexpected. This is a form of input data validation. If the information produced due to random generation is sent to other systems, this sanitizes and ensures unpredictability. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **Bad-function**  **(AUTOSAR.26.5.1A)** | Fully Checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 4.0 (prerelease) | Cert-msc50-cpp | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **BADFUNC.RANDOM.RAND** | Use of rand |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 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.

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| EXP-050-CPP | Medium | Probable | Medium | P8 | L2 |
| INT-050-CPP | Medium | Unlikely | Medium | P4 | L3 |
| OOP-050-CPP | Low | Unlikely | Medium | P2 | L3 |
| STR-002-C | High | Likely | Medium | P18 | L1 |
| MEM-052-CPP | High | Likely | Medium | P18 | L1 |
| DCL-003-C | Low | Unlikely | High | P1 | L3 |
| ERR-051-CPP | Low | Probable | Medium | P4 | L3 |
| FIO-050-CPP | Low | Likely | Medium | P6 | L2 |
| CON-051-CPP | Low | Probable | Low | P6 | L2 |
| MSC-050-CPP | Medium | Unlikely | Low | P6 | L2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Applies to all stored data on a system. This policy applies whenever there is sensitive data that is stored on the system or device. Sensitive data should be encrypted, the encryption keys should be stored separate from the encrypted data, there should be regular key rotation, and backup data must maintain the same levels of encryption. |
| Encryption in flight | Applies to all data that is being transmitted between systems, meaning that it must be applied any time there is any sort network communications with sensitive data. API endpoints should enforce HTTPS, legacy protocols such as SSL and old TLS versions should be avoided, and communications should use current versions of TLS. |
| Encryption in use | Applies to data being actively processed in data processing operations. Memory encryption should be used, and trusted execution environments should be used as well. Any encryption keys in memory must be protected and sensitive data should be cleared from memory immediately after being used. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process of proving a user’s identity. Basically, the log in screen and all its fun extra steps. This includes multi-factor authentication, password complexity requirements, session tokens, limiting login attempts, etc. |
| Authorization | The process of defining a user’s access rights. This is how a customer/user of a system can be separated from an administrator, for example. This ensures that Role-based access control and principles of least privilege are enforced. It allows users to get the exact level of permissions they need in order to interact with a system. |
| Accounting | The process of tracking a system and user’s activities. This allows for monitoring and handling security incidents as they occur. Both successful and failed authentication attempts should be logged in a secure location. Any authorization changes should be recorded. System access logs should also be retained. This information should be monitored and reviewed regularly to ensure nothing squirrely is going on. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [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 |