**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 | Debatably the most important security principle, involves ensuring that users aren’t inputting malicious content. Filters out most security issues. |
| 1. Heed Compiler Warnings | Modify your code in compliance with the warnings your compiler outputs. A program can still run with certain warnings, but they still represent a possible vulnerability that a tech-savvy hacker could exploit. |
| 1. Architect and Design for Security Policies | When designing software architecture, make sure that you enforce proper security policies. This prevents others from creating vulnerable code in the future. |
| 1. Keep It Simple | Complexity creates chaos. If it’s difficult to understand, it’s easy to accidentally create a weakness. By keeping it simple, you significantly lessen the chances of making an unknown error. |
| 1. Default Deny | Deny all access unless specifically allowed. This makes it so that a new user doesn’t have immediate access to data, unless an admin allows it. This ensures that an overstep doesn’t result in sensitive data being available to an unauthorized user. |
| 1. Adhere to the Principle of Least Privilege | Only give users the minimum access necessary to complete their given task. If someone has more power than they need, problems can occur. For example, they may be tempted to complete tasks they have not been properly trained on. Or even worse, they might not understand how much privilege they have, and might not properly secure their very powerful computer. |
| 1. Sanitize Data Sent to Other Systems | Just because data doesn’t harm some systems, doesn’t necessarily mean it won’t harm others. By cleaning data before transit, numerous possible security risks are avoided. |
| 1. Practice Defense in Depth | Multiple, overlapping layers of security provide the best possible protection for a system. There is no one way to fully protect a system (not that a system can ever be considered fully protected), but having multiple levels of security gives it its best chance. |
| 1. Use Effective Quality Assurance Techniques | Be specific and thorough when determining if a system meets its requirements. A system can only be considered complete if it explicitly completes each and every one of its goals to the fullest. |
| 1. Adopt a Secure Coding Standard | In most cases, a computer system is used by multiple individuals, each of which may use the system for a different purpose. Regardless, it is imperative that everyone follows strict rules and guidelines to properly prevent security vulnerabilities, such as never leaving their computer both unlocked and unattended. |

### 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] | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| Two different translation units define a class of the same name with differing definitions. Both create a class with one variable of the same name, but not with the same sequence of tokens. |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| The solution is to use a header file to introduce the object into both translation units. |
| // S.h  struct S {  int a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):** Head Compiler Warnings (this is definitely something the compiler would warn you about) and Keep It Simple (the compliant code above is much easier to understand). |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL60** | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.DEF.FDH LANG.STRUCT.DEF.ODH** | Function defined in header file Object defined in header file |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++1067, C++1509, C++1510** | N/A |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| Neglected to account for the case where number is equal to 0, which could make it uninitialized. |
| void set\_flag(int number, int \*sign\_flag) {  if (NULL == sign\_flag) {  return;  }    if (number > 0) {  \*sign\_flag = 1;  } else if (number < 0) {  \*sign\_flag = -1;  }  }    int is\_negative(int number) {  int sign;  set\_flag(number, &sign);  return sign < 0;  } |

| **Compliant Code** |
| --- |
| Accounts for the possibility of number being equal to 0. |
| void set\_flag(int number, int \*sign\_flag) {  if (NULL == sign\_flag) {  return;  }    /\* Account for number being 0 \*/  if (number >= 0) {  \*sign\_flag = 1;  } else {  \*sign\_flag = -1;  }  }    int is\_negative(int number) {  int sign = 0; /\* Initialize for defense-in-depth \*/  set\_flag(number, &sign);  return sign < 0;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques (uninitialized memory may make system not meet goal). |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | **uninitialized-local-read**  **uninitialized-variable-use** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-EXP33** | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.MEM.UVAR** | Uninitialized variable |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) | N/A | N/A | Automatically detects simple violations of this rule, although it may return some false positives. It may not catch more complex violations, such as initialization within functions taking uninitialized variables as arguments. It does catch the second noncompliant code example, and can be extended to catch the first as well |

#### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| Object is created from the results of a call to std::getenv(), which could return a null pointer and thus cause unintended behavior. |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| The results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    void f() {  const char \*tmpPtrVal = std::getenv("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques (null string may make system not meet goal). |
| --- |

**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=222953724) | 22.10 | **assert\_failure** | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF4770, DF4771, DF4772, DF4773, DF4774** | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | **NPD.CHECK.CALL.MIGHT** **NPD.CHECK.CALL.MUST** **NPD.CHECK.MIGHT** **NPD.CHECK.MUST** **NPD.CONST.CALL** **NPD.CONST.DEREF** **NPD.FUNC.CALL.MIGHT** **NPD.FUNC.CALL.MUST** **NPD.FUNC.MIGHT** **NPD.FUNC.MUST** **NPD.GEN.CALL.MIGHT** **NPD.GEN.CALL.MUST** **NPD.GEN.MIGHT** **NPD.GEN.MUST** **RNPD.CALL** **RNPD.DEREF** | N/A |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004- CPP] | Prevent SQL Injection |

| **Noncompliant Code** |
| --- |
| Uncleaned variable “username” could allow user to input something like “validuser' OR '1'='1”, which would cause SQL injection. |
| String sqlString = "SELECT \* FROM db\_user WHERE username = '"  + username +  "' AND password = '" + pwd + "'"; |

| **Compliant Code** |
| --- |
| PreparedStatement sanitizes the untrusted data before it is used in a query. |
| PreparedStatement stmt = connection.prepareStatement(sqlString); |

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

| **Principles(s):** Validate Input Data (this prevents user from giving malicious input) and Architect and Design for Security Policies (this will help prevent future developers from making this mistake in the future). |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [The Checker Framework](https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework) | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005- CPP] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| Variable ‘p’ is freed before p->next is executed, so it is read before it is freed. |
| #include <stdlib.h>    struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  for (struct node \*p = head; p != NULL; p = p->next) {  free(p);  }  } |

| **Compliant Code** |
| --- |
| p->next is called before p is freed, so freed memory is not read. |
| #include <stdlib.h>    **struct** node {  **int** value;  **struct** node \*next;  };    **void** free\_list(**struct** node \*head) {  **struct** node \*q;  **for** (**struct** node \*p = head; p != NULL; p = q) {      q = p->next;  **free**(p);    }  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques (accessing freed memory may make system not meet goal). |
| --- |

**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) | 22.04 | **dangling\_pointer\_use** | Supported  Astrée reports all accesses to freed allocated memory. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM30** | Detects memory accesses after its deallocation and double memory deallocations |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **ALLOC.UAF** | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **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-LLL] | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| Assert() is used to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
| #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** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution. |
| **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** Keep it simple (using Assert() on something so simple could complicate the code) and Use Effective Quality Assurance Techniques (this is a much more effective way to test the code). |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| DCL03-C | Low | Unlikely | High | **P1** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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) | 7.3p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) | N/A | N/A | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [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** | [STD-007- CPP] | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| Calling f() may result in a call to std::terminate() because throwing\_func() may throw an exception. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

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

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

| **Principles(s):** Head Compiler Warnings(you would definitely be warned about this) and Architect and Design for Security Policies (being able to abruptly stop the program could give malicious users an opening). |
| --- |

**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 | **stdlib-use** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **BADFUNC.ABORT BADFUNC.EXIT** | Use of abort Use of exit |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.24 | [**V667**](https://pvs-studio.com/en/docs/warnings/v667/)**,**[**V2014**](https://pvs-studio.com/en/docs/warnings/v2014/) | N/A |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stdlib-use** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-008- CPP] | Copy operations must not mutate the source object |

| **Noncompliant Code** |
| --- |
| The copy operations for A mutate the source operand by resetting its member variable m to 0. |
| #include <algorithm>  #include <vector>    class A {  mutable int m;    public:  A() : m(0) {}  explicit A(int m) : m(m) {}    A(const A &other) : m(other.m) {  other.m = 0;  }    A& operator=(const A &other) {  if (&other != this) {  m = other.m;  other.m = 0;  }  return \*this;  }    int get\_m() const { return m; }  };    void f() {  std::vector<A> v{10};  A obj(12);  std::fill(v.begin(), v.end(), obj);  } |

| **Compliant Code** |
| --- |
| The copy operations for A no longer mutate the source operand. |
| #include <algorithm>  #include <vector>    class A {  int m;    public:  A() : m(0) {}  explicit A(int m) : m(m) {}    A(const A &other) : m(other.m) {}  A(A &&other) : m(other.m) { other.m = 0; }    A& operator=(const A &other) {  if (&other != this) {  m = other.m;  }  return \*this;  }    A& operator=(A &&other) {  m = other.m;  other.m = 0;  return \*this;  }    int get\_m() const { return m; }  };    void f() {  std::vector<A> v{10};  A obj(12);  std::fill(v.begin(), v.end(), obj);  } |

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

| **Principles(s):** Keep it Simple (mutating the original source is both bad practice and very complicated to work with) and Heed Compiler Warnings (you will be warned of this). |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| OOP58-CPP | Low | Likely | Low | **P9** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.FUNCS.COPINC** | Copy Operation Parameter Is Not const |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4075** | N/A |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2023.1 | **CERT.OOP.COPY\_MUTATES** | N/A |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-OOP58-a** | Copy operations must not mutate the source object |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-009- CPP] | Guarantee that container indices and iterators are within the valid range |

| **Noncompliant Code** |
| --- |
| Because pos is declared as a (signed) int, this parameter can assume a negative value, resulting in a write outside the bounds of the memory referenced by table. |
| #include <cstddef>    **void** insert\_in\_table(**int** \*table, std::**size\_t** tableSize, **int** pos, **int** value) {  **if** (pos >= tableSize) {      // Handle error  **return**;    }    table[pos] = value;  } |

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

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

| **Principles(s):** Validate Input Data (prevent the user from being able to input a negative value) and Keep It Simple (by simply preventing the user from doing this, much less code is needed for security purposes). |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| CTR50-CPP | High | Likely | High | **P9** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **overflow\_upon\_dereference** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.MEM.BO LANG.MEM.BU LANG.MEM.TO LANG.MEM.TU LANG.MEM.TBA LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.STRUCT.PARITH** | Buffer overrun Buffer underrun Type overrun Type underrun Tainted buffer access Pointer before beginning of object Pointer past end of object Pointer Arithmetic |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-CTR50-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: CTR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr50cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010- CPP] | Call functions with the correct number and type of arguments |

| **Noncompliant Code** |
| --- |
| Attempts to take the base-2 logarithm of a complex number, resulting in undefined behavior. |
| #include <tgmath.h>    **void** func(**void**) {  **double** complex c = 2.0 + 4.0 \* I;  **double** complex result = log2(c);  } |

| **Compliant Code** |
| --- |
| If the clog2() function is not available for an implementation as an extension, the programmer can take the base-2 logarithm of a complex number, using log() instead of log2(), because log() can be used on complex arguments. |
| #include <tgmath.h>    void func(void) {  double complex c = 2.0 + 4.0 \* I;  double complex result = log(c)/log(2);  } |

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

| **Principles(s):** Keep It Simple (incorrect arguments like this can lead to confusing errors) and Head Complier Warnings (this would come up as a warning, possibly not an error, at least until it was run). |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | **incompatible-argument-type**  **parameter-match**  **parameter-match-computed**  **parameter-match-type** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-EXP37** | N/A |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.FUNCS.APM** | Array parameter mismatch |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this rule. In particular, it ensures that all calls to open() supply exactly two arguments if the second argument does not involve O\_CREAT, and exactly three arguments if the second argument does involve O\_CREAT |

### 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 |
| STD-002-CPP | High | Probable | Medium | **P12** | **L1** |
| STD-003-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-004-CPP | High | Probable | Medium | **P12** | **L1** |
| STD-005-CPP | High | Likely | Medium | **P18** | **L1** |
| STD-006-CPP | DCL03-C | Low | Unlikely | High | **P1** |
| STD-007-CPP | Low | Probable | Medium | **P4** | **L3** |
| STD-008-CPP | OOP58-CPP | Low | Likely | Low | **P9** |
| STD-009-CPP | CTR50-CPP | High | Likely | High | **P9** |
| STD-010-CPP | Medium | Probable | High | **P4** | **L3** |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption in rest secures stored data. This mainly includes where and how the data is left unattended, such as within a database. Various encryption tools can protect data at rest, but various security policies (such as defense in depth) can greatly aid in protecting it. |
| Encryption at flight | Encryption at flight secures moving data. This may include data moving from one database to the next, between devices, etc. Various security tools can assist with this, but the main policy to be aware of here is to Sanitize Data Sent to Other Systems. This ensures that both the data and the new system is as secure as possible. |
| Encryption in use | Encryption in use secures data as it is being analyzed, added to, removed from, etc. Securing data in use is best done by utilizing proper security protocols in regards to authorization and authentication. A great example here is the principle of least privilege, where each user is only given the minimum level of access they need to complete their own job. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of confirming if someone is who they say they are. This is most often done via a username and password. Though this does not make it impossible for a malicious user to access the data by any means, it is one piece of the defense in depth coverage, and can prevent numerous types of attacks. |
| Authorization | Authorization is the process of determining and specifying the level of access each user is allowed. This is commonly done via the Principle of Least Privilege, which dictates that each user is only given the level of access that is necessary to complete their specific job, and nothing more. |
| Accounting | Accounting is the process of recording and tracking who used the system, as well as when and how. Though this doesn’t necessarily prevent attacks, it can make it much easier to locate the source of attacks, figure out how the attacker bypassed the security, and patch it immediately. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 4/7/2023 | Project One | Jason Veno | Aaron Demory |
| [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 |