**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 2](#_Toc52464058)

[Coding Standard 1 2](#_Toc52464059)

[Coding Standard 2 2](#_Toc52464060)

[Coding Standard 3 2](#_Toc52464061)

[Coding Standard 4 2](#_Toc52464062)

[Coding Standard 5 2](#_Toc52464063)

[Coding Standard 6 2](#_Toc52464064)

[Coding Standard 7 2](#_Toc52464065)

[Coding Standard 8 2](#_Toc52464066)

[Coding Standard 9 2](#_Toc52464067)

[Coding Standard 10 2](#_Toc52464068)

[Defense-in-Depth Illustration 2](#_Toc52464069)

[Project One 2](#_Toc52464070)

[1. Revise the C/C++ Standards 2](#_Toc52464071)

[2. Risk Assessment 2](#_Toc52464072)

[3. Automated Detection 2](#_Toc52464073)

[4. Automation 2](#_Toc52464074)

[5. Summary of Risk Assessments 2](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 2](#_Toc52464076)

[7. Map the Principles 2](#_Toc52464077)

[Audit Controls and Management 2](#_Toc52464078)

[Enforcement 2](#_Toc52464079)

[Exceptions Process 2](#_Toc52464080)

[Distribution 2](#_Toc52464081)

[Policy Change Control 2](#_Toc52464082)

[Policy Version History 2](#_Toc52464083)

[Appendix A Lookups 2](#_Toc52464084)

[Approved C/C++ Language Acronyms 2](#_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 | User data must be validating to ensure that malicious actors are not able to inject harmful strings into the system, either SQL injection or possible Remote Code Execution exploits. Preventing this at the time of input rather than further down the line ensure that the data is secure before it gets sent further into the system. |
| 1. Heed Compiler Warnings | Compiler warnings can serve as one of the earliest layers of defense in depth protection. They can alert the programming to code that is unsafe or vulnerable as well as if the code is depreciated. |
| 1. Architect and Design for Security Policies | Security should be a constant though in the mind of developers. They should plan their projects with security in mind from the drawing board and make it an integral part of the overall architecture of the program. |
| 1. Keep It Simple | The more complicated your security procedures are the more likely the end user is going to attempt to work around them rather than work with them. This is counter-productive and sets up an antagonistic relationship as well as an unsecure system. Security should comprehensive but simple to use. |
| 1. Default Deny | When authenticating users the system should default to denying them access rather and being given higher access as needed, as opposed to users having all access and needing to be manually removed/blocked from parts of the system they shouldn’t have access too. This makes for a much more secure system as if a hacker does get access to an account or even worse creates a new one, they only have minimal permissions to access vital systems. |
| 1. Adhere to the Principle of Least Privilege | Users and systems should have the least amount of privileges possible to complete their task and never more than they need. This helps prevent attacks by locking down the privileges and being able to monitor attack vectors more closely. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data means to clean, filter, or modify data to ensure that it is safe and secure before sending it to another system to be used. |
| 1. Practice Defense in Depth | This is the practice of having multiple layers of security to provide more protection than just 1 security layer. This ensures that even if one layer of security is compromised that the system remains secure. |
| 1. Use Effective Quality Assurance Techniques | Quality Assurance and testing is very important to secure code. This can be the last line of defense before releasing into production so using effective techniques and tests for security vulnerabilities is vital. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard sets the precedence and requirements for the developers to follow to create safe and secure code. This is important as on larger teams miscommunication and varying ways to go about security can lead to vulnerabilities whereas with a standard to work from the team is always on the same page. |

### 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] | Do not modify the standard namespace. The standard library introduces the namespace std for standards-provided declarations such as std::string, std::vector, and std::for\_each. However, it is undefined behavior to introduce new declarations in namespace std except under special circumstances. |

| **Noncompliant Code** |
| --- |
| This block adds the integer x to the std namespace which is undefined behavior. |
| namespace std {  int x = 5;  } |

| **Compliant Code** |
| --- |
| The integer x is placed into a new namespace without conflicts. |
| namespace project {  int x = 5;  } |

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

| **Principles(s):** Architect and Design for Security Policies by avoiding modifying the standard namespace you are making sure your architecture is secure and avoid undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Ensure that operations on signed integers do not result in overflow. |

| **Noncompliant Code** |
| --- |
| This code can result in overflow for sufficiently large or sufficiently small numbers. |
| int add(int x, int y) {  return x + y;  } |

| **Compliant Code** |
| --- |
| This code checks the result to ensure it doesn’t overflow and throws an exception if it does. |
| int add(int x, int y) {  if((y > 0 && (x > INT\_MAX – y)) ||  (y < 0 && (x < INT\_MIN – y)) {  throw std::overflow\_exception;  }  return x + y;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques: By using effective QA techniques these errors will be caught and prevent errors down the line with the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **integer-overflow** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **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 |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **TAINTED\_SCALAR**  **BAD\_SHIFT** | Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-INT32-a** **CERT\_C-INT32-b** **CERT\_C-INT32-c** | Avoid signed integer overflows Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Range check element access |

| **Noncompliant Code** |
| --- |
| This for this block the value retuned by get\_index may be greater than the size of the string which results in undefined behavior. |
| extern std::size\_t get\_index();    void foo() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| By using a try/catch block we can handle the error. |
| extern std::size\_t get\_index();    void f() {    std::string s("01234567");    try {      s.at(get\_index()) = '1';    } catch (std::out\_of\_range &) {      // Handle error    }  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques: By using effective QA techniques these errors will be caught and prevent errors down the line with the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | **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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-STR53-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++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Sanitize and tokenize all input to check for potential SQL injection before inputting the query to the database. |

| **Noncompliant Code** |
| --- |
| This code passes the string directly to the database without checking it first. |
| void sqlQuery(std::string sql) {  database.query(sql);  } |

| **Compliant Code** |
| --- |
| This code passes the query to a separate function to check if it’s a potential injection vector and only passes it to the database if it is not. |
| bool isSafeQuery(std::string query) {  // confirm query is safe  return safe;  }  void sqlQuery(std::string sql) {  if (!isSafeQuery(sql)) {  throw std::exception(“Possible SQL injection”);  }  database.query(sql);  } |

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

| **Principles(s):** Validate Input Data: By validating input data we can ensure that no injection is possible. Sanitize Data Sent to Other Systems: Sanitizing the data being sent from the input before it goes to other systems is an extra step that will help prevent SQL injection attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft SOAtest](https://www.parasoft.com/products/parasoft-soatest/) | 1.0 | Automatic | Automatically checks code for SQL vulerabilites |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| This code declares a Foo point then deletes it, after it is deleted it attempts to access foo which has been freed from memory. |
| struct Foo {  int bar;  };  void nonCompliant() {  Foo\* foo = new Foo;  delete foo;  foo->bar;  } |

| **Compliant Code** |
| --- |
| This code uses the automatic storage duration and allows foo to drop out of memory automatically at the end of the function as it falls out of scope. |
| struct Foo {  int bar;  };  void compliant() {  Foo\* foo = new Foo;  foo->bar;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: by adopting a secure coding standard we can ensure that developers only access memory safely. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | **ALLOC.UAF** | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Make assert messages clear to avoid confusion and make debugging easier. |

| **Noncompliant Code** |
| --- |
| This block has a static assertion with a vague message |
| void foo() {  static\_assert(2 + 2 == 4, "something happened");  } |

| **Compliant Code** |
| --- |
| This block has the same static assertion with a much more clear message |
| void foo() {  static\_assert(2 + 2 == 4, "2 + 2 = 4");  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques by making the messages descriptive it becomes much easier to diagnose issues. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | Low | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [sonarLint](https://www.sonarsource.com/products/sonarlint/) | 2.7 | MISRA C:2012 | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| No where along the chain of calls does the thrown exception get checked. |
| void throwingFunction() {  throw std::runtime\_error("catch this");  }  void foo() {  throwingFunction();  }  int main() {  foo();  } |

| **Compliant Code** |
| --- |
| The main function handles the exception by surrounding the function call to foo() with a try/catch block. |
| void throwingFunction() {  throw std::runtime\_error("catch this");  }  void foo() {  throwingFunction();  }  int main() {  try {  foo();  }  catch (std::runtime\_error const& e) {  e.what();  }  } |

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

| **Principles(s):** Architect and Design for Security Policies, Adopt a Secure Coding Standard  By following these two principles we can ensure that all possible exceptions get handled and prevent the program from failing ungracefully. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | 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** |
| --- | --- | --- |
| Containers | [STD-008-CPP] | Guarantee that container indices and iterators are within the valid range |

| **Noncompliant Code** |
| --- |
| This code fails to check that an int can be < 0 and as such would fail for all negative numbers. |
| void insertInArray(int\* list, std::size\_t arraySize, int pos, int value) {  if (pos >= arraySize) {  throw std::runtime\_error("Position is out of range of array index");  }  list[pos] = value;  } |

| **Compliant Code** |
| --- |
| This code correctly checks against negative numbers. |
| void insertInArray(int\* list, std::size\_t arraySize, int pos, int value) {  if (pos < 0 || pos >= arraySize) {  throw std::runtime\_error("Position is out of range of array index");  }  list[pos] = value;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques & Architect and Design for Security Policies  These 2 principles ensure that these errors are caught and designing a system from the ground up to avoid this kind of vulnerability is the way to ensure that it doesn’t happen without incurring a large cost if it’s discovered too late. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **ALLOC.SIZE.IOFLOW** **ALLOC.SIZE.MULOFLOW** **MISC.MEM.SIZE.BAD** | Integer Overflow of Allocation Size Multiplication Overflow of Allocation Size Unreasonable Size Argument |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **REVERSE\_NEGATIVE** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **621 S** | Enhanced enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-ARR32-a** | Ensure the size of the variable length array is in valid range |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Functions | [STD-009-CPP] | Value-returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| This function only works when foo is a negative number and if you pass a positive value you get undefined behavior. |
| int absoluteValue(int foo) {  if (foo < 0) {  return -foo;  }  } |

| **Compliant Code** |
| --- |
| This code has a return statement for every exit path so it will work correctly with both positive and negative values of foo. |
| int absoluteValue(int foo) {  if (foo < 0) {  return -foo;  }  return foo;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques & Architect and Design for Security Policies  These 2 principles ensure that these errors are caught and designing a system from the ground up to avoid this kind of vulnerability is the way to ensure that it doesn’t happen without incurring a large cost if it’s discovered too late. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | 8 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | **LANG.STRUCT.MRS** | Missing return statement |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **2 D, 36 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MSC52-a** | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | [STD-010-CPP] | Close files when they are no longer needed. |

| **Noncompliant Code** |
| --- |
| This block opens a file and returns from the function however it never closes the file. |
| void fileHandler(std::string& fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  throw std::runtime\_error("File cannot be opened");  }  std::terminate();  } |

| **Compliant Code** |
| --- |
| This block correctly ensures that the file is closed before calling terminate. |
| void fileHandler(std::string& fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  throw std::runtime\_error("File cannot be opened");  }  file.close();  if (file.fail()) {  throw std::runtime\_error("File failed to close");  }  std::terminate();    } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques & Architect and Design for Security Policies  These 2 principles ensure that these errors are caught and designing a system from the ground up to avoid this kind of vulnerability is the way to ensure that it doesn’t happen without incurring a large cost if it’s discovered too late. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | **ALLOC.LEAK** | Leak |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

### 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.

We will add automation to the build, design, and verify and test stages of pre-production, by running our code through the automated tools in this step we should be able to catch most if not issues that we are actively looking for in the code. We will also add it to the Maintaince and Response stages of production that will ensure that we deliver update to the software that are just as robust and secure as the original release of the system.

### 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 | Medium | 2 |
| **STD-002-CPP** | High | Likely | High | Medium | 2 |
| **STD-003-CPP** | High | Unlikely | Medium | Medium | 2 |
| **STD-004-CPP** | High | Likely | High | High | 1 |
| **STD-005-CPP** | High | Likely | Medium | High | 1 |
| **STD-006-CPP** | Low | Likely | Medium | Low | 4 |
| **STD-007-CPP** | Low | Probable | Medium | Low | 3 |
| **STD-008-CPP** | High | Probable | High | Medium | 2 |
| **STD-009-CPP** | Medium | Probable | Medium | Medium | 2 |
| **STD-010-CPP** | Medium | Unlikely | Medium | Low | 3 |

### 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 is encryption used to protect data that is stored on a disk or other back up media. We will use this to protect the database of passwords and other store information. This can be done through salting and hashing the data or by requiring a secure key to access it. |
| Encryption at flight | This is encryption used to protect data as it is being transferred between locations, usually server to client or vice versa. This is achieved through various measures such as checksum verifications, private/public keys and session keys. |
| Encryption in use | This is encryption meant to protect the data as it’s being used by its intended recipient. This can be achieved through multi-factor authentication or single sign on systems. It can be further reduced by following the principle of least privilege and to default deny ensuring that the user does not have access to further data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is having the user prove they are who they say they are, usually through a log-in system with a username and password. You can make this more secure by requiring Multi-Factor Authentication. New users must be added to the authentication service in order to access the system at all. |
| Authorization | Authorization is about giving access to users to access information. This means that there should be multiple layers of security access and privileges in order to be able to best define what each user should have access to. |
| Accounting | Accounting is about logging changes and accesses made to the system to be reviewed later. This should log all files accessed by users, all changes made to the database, and many other events. To be effective accounting just can’t be logged, it has to be reviewed otherwise you just create data for the sake of having it. |

**\***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** | 11/09/2023 | Added security principles and standards | Alex Wells |  |
| **1.2** | 11/30/2023 | [Insert text.] | Alex Wells |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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