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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_heading=h.gjdgxs)

[Purpose 2](#_heading=h.30j0zll)

[Scope 2](#_heading=h.1fob9te)

[Module Three Milestone 2](#_heading=h.3znysh7)

[Ten Core Security Principles 2](#_heading=h.4d34og8)

[C/C++ Ten Coding Standards 3](#_heading=h.2s8eyo1)

[Coding Standard 1 4](#_heading=h.3rdcrjn)

[Coding Standard 2 5](#_heading=h.26in1rg)

[Coding Standard 3 6](#_heading=h.lnxbz9)

[Coding Standard 4 7](#_heading=h.35nkun2)

[Coding Standard 5 8](#_heading=h.1ksv4uv)

[Coding Standard 6 9](#_heading=h.2jxsxqh)

[Coding Standard 7 10](#_heading=h.z337ya)

[Coding Standard 8 11](#_heading=h.28h4qwu)

[Coding Standard 9 13](#_heading=h.nmf14n)

[Coding Standard 10 14](#_heading=h.37m2jsg)

[Defense-in-Depth Illustration 15](#_heading=h.1mrcu09)

[Project One 15](#_heading=h.46r0co2)

[1.](#_heading=h.2lwamvv) Revise the C/C++ Standards 15

[2.](#_heading=h.111kx3o) Risk Assessment 15

[3.](#_heading=h.3l18frh) Automated Detection 15

[4.](#_heading=h.206ipza) Automation 15

[5.](#_heading=h.4k668n3) Summary of Risk Assessments 16

[6.](#_heading=h.2zbgiuw) Create Policies for Encryption and Triple A 16

[7.](#_heading=h.1egqt2p) Map the Principles 17

[Audit Controls and Management 18](#_heading=h.3ygebqi)

[Enforcement 18](#_heading=h.2dlolyb)

[Exceptions Process 18](#_heading=h.sqyw64)

[Distribution 19](#_heading=h.3cqmetx)

[Policy Change Control 19](#_heading=h.1rvwp1q)

[Policy Version History 19](#_heading=h.4bvk7pj)

[Appendix A Lookups 19](#_heading=h.2r0uhxc)

[Approved C/C++ Language Acronyms 19](#_heading=h.1664s55)

## 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 highlights the importance of treating all input data as data that is potentially malicious. Before utilizing data within an application, it should be validated with a set of validations that can check for things like expected format, data type, and length. This principle can help protect against things like SQL injection. |
| 1. Heed Compiler Warnings | The heed compiler warnings principle shows that compiler warnings should generally not be ignored, as they can be an early warning system for potential security vulnerabilities and logical errors in code. Warnings can be configured to show up as errors, which could encourage developers to fix them. |
| 1. Architect and Design for Security Policies | This principle emphasizes that security should be an integral part of the architectural and design phases of software development. Also, this can include designing systems to be resilient against known risks, trying to implement a secure connection, and protecting data integrity and confidentiality. |
| 1. Keep It Simple | This principle shows that complexity can negatively impact security, and that simplistic design and implementation, can reduce the likelihood of security vulnerabilities. Simpler systems can be easier to analyze, understand, and secure. So, by limiting the amount of components and interactions, the attack surface can be reduced, which should make the system more secure. |
| 1. Default Deny | This principle states that a system should deny all operations, transactions, or communications unless explicitly allowed by particular security policies. This approach can limit potential vulnerabilities by trying to ensure that only permitted actions can be run, which can reduce the risk of unauthorized access and or operations. |
| 1. Adhere to the Principle of Least Privilege | This principle states that users and systems should operate using the minimum set of privileges necessary to complete their tasks. Implementing this approach can limit the potential damage from accidents or potential breaches by limiting access and permissions to only what is strictly required, which, in turn, should contain breaches to a limited scope. |
| 1. Sanitize Data Sent to Other Systems | This principle involves cleaning and validating all data before it is sent to other systems, trying to ensure that the data does not contain anything malicious which could potentially be used to exploit potential vulnerabilities on a system. Data sanitization can help protect against SQL injection attacks and helps maintain the integrity and security of systems. |
| 1. Practice Defense in Depth | This principle involves implementing multiple layers of security controls and practices throughout a system. The goal of this principle is to set up a system with multiple layers so that if one security layer gets compromised, there are other layers that are still effective. Implementing this can involve a combination of physical controls like locking a room, technical controls, and administrative controls to help protect and secure a system. |
| 1. Use Effective Quality Assurance Techniques | This principle, quality assurance, can identify and fix potential security issues before software is released. Some quality assurance techniques can involve testing like code reviews, and automated testing, which can help with trying to ensure that security measures are correctly implemented and effective. This approach can be important for developing secure software. |
| 1. Adopt a Secure Coding Standard | This principle can provide guidelines and best practices for writing secure code. They can help developers avoid some potential common mistakes that could lead to vulnerabilities and can help try and ensure that security considerations are applied throughout the development process. Adopting and adhering to a secure coding standard can be important for building resilient and secure software applications. |

### 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 read uninitialized memory. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This rule shows the importance of initializing variables, and declaring their data types before their use, as this can help prevent unpredictable behavior and potential vulnerabilities. |

| **Noncompliant Code** |
| --- |
| In this code block, undefined behavior occurs when there is an attempt to print an uninitialized variable. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <iostream>    void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| In this code block, before the value of the variable is printed, the variable is initialized. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <iostream>    void f() {  int i = 0;  std::cout << i;  } |

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

| **Principles(s):** 9, 10. An example of a secure coding practice could potentially be initializing memory, which could help with preventing unpredictable behavior and vulnerabilities. Also, initializing variable could potentially be checked during code reviews and automated testing. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | Latest | Uninitialized Variables | This tool could help with detecting the use of variables that are not initialized before their use. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not depend on the order of evaluation for side effects. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This is applicable as the unpredictability in the evaluation order of expressions with side effects can lead to unidentified behavior and inaccurate data values. |

| **Noncompliant Code** |
| --- |
| In this code block, the behavior of the expression is undefined, because the i variable is being evaluated more than once in an unsequenced manner. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| void f(int i, const int \*b) {  int a = i + b[++i];  // ...  } |

| **Compliant Code** |
| --- |
| In this code block, the examples can only be interpreted one way, and are independent of the order of operation. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| void f(int i, const int \*b) {  ++i;  int a = i + b[i];  // ...  }  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):** 4. Creating expressions that could potentially only be interpreted one way could reduce complexity. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | Latest | Data Flow Analysis | This tool could help analyze the flow of data for applications to monitor if operations are secure. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This rule can help with trying to ensure that strings have adequate space including the null terminator, which can be important for preventing buffer overflows. |

| **Noncompliant Code** |
| --- |
| In this code block, the code could lead to buffer overflow due to the input being unbound. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| In this code block, std::string can be used to try and make sure that data is not truncated, and to help with buffer overflow protection. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <iostream>  #include <string>    void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** 1. Creating adequate space for strings, including null terminators, could support input validation by preventing buffer overflows. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | Latest | Buffer Overflow | This tool could help detect potential buffer overflows by checking bounds on string operations. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Do not attempt to create a std::string from a null pointer. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This rule can help support practices that limit SQL Injection risks by ensuring that string operations are performed correctly. |

| **Noncompliant Code** |
| --- |
| In this code block, std::getenv() is called, and this results in a std::string object being created. However, because std::getenv() returns a null pointer on failure, this could lead to no environment variable, which can lead to undefined behavior. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In this code block, before the std::string object is created, the results from the call to std::getenv() are checked for null. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #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):** 1. Proper string handling practices could help limit risks like SQL injection, which could be a result of improperly handled input data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify Static Code Analyzer | Latest | SQL Injection | This tool could help identify code patterns that could potentially lead to SQL injection vulnerabilities. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Do not access freed memory. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This rule can help with memory protection by trying to prevent accessing invalid memory areas, which can be important for maintaining the integrity and security of the program’s memory. |

| **Noncompliant Code** |
| --- |
| In this code block, after being deallocated, s is dereferenced. If the access results in a write-after-free, arbitrary code could be run if the vulnerability is exploited. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this code block, deallocation of the dynamically allocated memory does not occur until it is no longer required. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):** 5. By not accessing freed memory, the code could be set up where only validated actions are permitted. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| AddressSanitizer | Latest | Memory Errors | This tool could help detect memory errors by catching out of bounds and use after free bugs. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Value-returning functions must return a value from all exit paths. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This rule can help with trying to ensure that function behave as expected, which can help with assertions by verifying assumptions in code like function behavior. |

| **Noncompliant Code** |
| --- |
| In this code block, not all code paths return a value, due to the developer did not return the input value for positive input. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| In this code block, a value is returned for all the code paths. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  return a;  } |

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

| **Principles(s):** 9. Testing that all code paths within a function return a value could potentially be mapped to quality assurance practices. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | Latest | Control Flow | This tool could help analyze control flow to check if assertions are consistently applied. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle all exceptions. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This rule can help exception handling by trying to ensure that all exception are caught and handled appropriately, which can help prevent unexpected program behavior. |

| **Noncompliant Code** |
| --- |
| In this code block, throwing\_func throws exceptions, and both f() and main() do not catch those exceptions. No matching handler is found for the thrown exception, so std::terminate() is called. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this code block, expected management of external resources occurs due to the main entry point handling all exceptions, which can help with trying to ensure that the stack is unwound up to the main() function. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| 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):** 8. Proper exception handling could involve having multiple check, so if one falls, there are others to handle the exception. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | Latest | Exception Safety | This tool could help check for code that might throw exceptions and could help with properly handling exceptions. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Concurrent Safety | STD-008-CPP | Prevent data races when accessing bit-fields from multiple threads. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This rule can be important for concurrent safety as it can help with trying to ensure that access to shared data structures is properly synchronized, which can help with preventing data races that can lead to unpredictable behavior. |

| **Noncompliant Code** |
| --- |
| In this code block, single memory location could be used as storage for adjacent bit-fields. So, undefined behaviour could result by modifying adjacent bit-fields in different threads. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| struct MultiThreadedFlags {  unsigned int flag1 : 2;  unsigned int flag2 : 2;  };    MultiThreadedFlags flags;    void thread1() {  flags.flag1 = 1;  }    void thread2() {  flags.flag2 = 2;  } |

| **Compliant Code** |
| --- |
| In this code block, data races are prevented by protecting all accesses of the flags with mutex. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <mutex>    struct MultiThreadedFlags {  unsigned int flag1 : 2;  unsigned int flag2 : 2;  };    struct MtfMutex {  MultiThreadedFlags s;  std::mutex mutex;  };    MtfMutex flags;    void thread1() {  std::lock\_guard<std::mutex> lk(flags.mutex);  flags.s.flag1 = 1;  }    void thread2() {  std::lock\_guard<std::mutex> lk(flags.mutex);  flags.s.flag2 = 2;  } |

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

| **Principles(s):** Synchronizing access to shared data structures could help with preventing data races, which could potentially help with maintaining stability across multiple security layers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Possible | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ThreadSanitizer | Latest | Thread Safety | This tool could help detect concurrency errors by identifying things like data races for multithreaded applications. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Resource Management | STD-009-CPP | Properly deallocate dynamically allocated resources. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This rule can help prevent resource leaks and promote efficient use of system resources by trying to ensure that dynamic resources are managed correctly. |

| **Noncompliant Code** |
| --- |
| In this code block, the placement operator gets the local variable space as an expression. After the call, the resulting pointer is passed to ::operator delete(), which attempts to free memory that :: operator new() did not return, resulting in undefined behavior. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    delete s1;  } |

| **Compliant Code** |
| --- |
| In this code block, the use of ::operator delete() is removed, and the s1’s destructor is called directly. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    s1->~S();  } |

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

| **Principles(s):** 4. Efficient resource management could help with reducing complexity. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LeakSanitizer | Latest | Leak Detection | This tool could help identify memory leaks. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input validation | STD-0010-CPP | Range check element access. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) This rule can help prevent accessing elements outside their bounds, which can be an important aspect of implementing secure input validation. |

| **Noncompliant Code** |
| --- |
| In this code block, undefined behavior results from the value returned by the call to get\_index() potentially being being greater than the number of elements stored in the string. (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <string>    extern std::size\_t get\_index();    void f() {  std::string s("01234567");  s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| In this code block, the std::basic\_string::at() function is used, which behaves similarly to the index operator[] but throws an std::out\_of\_range exception if the position is greater than or equal to size(). (SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence, n.d.) |
| #include <stdexcept>  #include <string>  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):** 1, 10. Range checking could potentially help test that all inputs are validated before being utilized, which could help with preventing errors and potential security risks. Adhering to input validation techniques could potentially be aligned with secure coding practices. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | Latest | Input Validation Checks | This tool could help check for common input validation errors. |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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.

To integrate security within the organization's framework, to create a DevSecOps pipeline with automation could be important. For the development phase, IDE plugins and pre commit hooks could help with automating security checks, to check for compliance with security standards. During continuous integration, tools like SonarQube and Fortify could help with automatically performing static and dynamic analyses to potentially catch vulnerabilities early. For deployment and operation, automated configuration management tools could help with meeting compliance during release, while Runtime Application Self-Protection could continue to monitor and defend applications. Feedback mechanisms, automated through tools like security orchestration and automated response, could continue to enhance security measures.

### 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 | Medium | Likely | Medium | Medium | 2 |
| STD-003-CPP | High | Possible | High | High | 1 |
| STD-004-CPP | High | Possible | High | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Medium | Unlikely | Low | Medium | 3 |
| STD-007-CPP | High | Possible | High | High | 1 |
| STD-008-CPP | Medium | Possible | Medium | Medium | 2 |
| STD-009-CPP | High | Possible | High | High | 1 |
| STD-010-CPP | High | Likely | Medium | High | 1 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | Encryption at rest could involve encrypting stored data. This policy could apply to all data stored on organizational servers, cloud storage, and backup systems. Encryption at rest can be important for preventing unauthorized access to data. |
| Encryption in flight | Encryption in flight could involve encrypting data that is being transferred over networks. This policy could be applied to all data transfers within and outside the organization’s network, including data shared utilizing APIs, web services, and email communications. This policy could be important for protecting data during transit. |
| Encryption in use | Encryption in use could involve encrypting data that might be actively being utilized by applications. This policy could cover all data in the process of being utilized by an application within the organization. Encryption in use should help with protecting data from unauthorized access and or leaks during operations. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication could involve verifying the identity of a user, process, and or device, often as a prerequisite to granting access to system resources. This policy could apply to all system logins and application accesses. Authentication could help with trying to only granting access to systems and data to authorized individuals, which can help with preventing unauthorized access. |
| Authorization | Authorization could involve trying to only grant access to authenticated users for specific resources and actions. This policy could apply to user roles and permissions for the system. Authorization could help grant access permissions based on job functions, which could potentially help limit the risk of unauthorized system access. |
| Accounting | Accounting could involve tracking and logging user activities for the purpose of analyzing resource utilization and potentially detecting anomalies. This policy could apply to all user activities, database changes, and potentially sensitive data access. Accounting could help with monitoring user actions, policy compliance, and potentially detecting security incidents. |

**\***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 | 3/24/2024 | 3-2 Milestone: Coding Standards | Nate Bennett |  |
| 1.2 | 4/14/2024 | 6-2 Project One: Security Policy | Nate Bennett |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

Works Cited

*SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard -*

*Confluence*. (n.d.). Wiki.sei.cmu.edu.

https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682