**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 | All external data should be validated to prevent unintended activity in a system. Attacks like buffer overflows, SQL injection, and cross-site scripting can occur when malicious users pass carefully formatted data into a system. By ensuring that only the intended data can be passed into a system, these sorts of exploits can be eliminated. |
| 1. Heed Compiler Warnings | When compiling code, it is important to pay attention to any warnings that are generated. While the code will still compile with these warnings, they indicate problems with the code that could lead to issues in the future. When these warnings arise, developers need to edit the code to eliminate them. |
| 1. Architect and Design for Security Policies | Design software, from the start, to enforce security policies. This could include segmenting a system into smaller subsystems so that resources are only available to users with certain privileges. |
| 1. Keep It Simple | Keep the design of a system as simple as possible. Overly complex systems have many more avenues for things to go wrong when using it or setting it up and are harder to maintain. Simple designs are easier to thoroughly test as well, adding an extra layer of assurance. |
| 1. Default Deny | This principle means that access to resources is denied by default. Only users explicitly granted permission can access these resources. This is much easier to implement that an exclusion-based policy that seeks to control access by allowing everyone but explicitly blocked users from accessing a resource. |
| 1. Adhere to the Principle of Least Privilege | Users and processes should only have the bare minimum privileges to carry out their tasks. This can help eliminate exploits made possible by running in an environment with elevated privileges. We certainly would not want a user of a website to operate with the same permissions as the site’s administrator! |
| 1. Sanitize Data Sent to Other Systems | When sending data to subsystems, it is important for the calling process to clean up the data. These subsystems, like a SQL database or a command prompt, do not understand the context of the data passed to it and would execute any valid command sent to it, whether it was intended functionality of the system or not. Unneeded features should also be deactivated so that they are not used to exploit the system. |
| 1. Practice Defense in Depth | It is important to have several layers of defense. There are many methods to secure a system, but each has its weaknesses. Ideally, multiple layers would cover each other’s weaknesses, provide redundancy, resulting in a secure system. |
| 1. Use Effective Quality Assurance Techniques | The implementation of testing techniques can be useful in identifying vulnerabilities in a system. Code reviews, penetration testing, and fuzz testing, an automated method of injecting random or invalid data, can all contribute to a well-rounded quality assurance program. The use of outside reviewers can also help examine systems from a different perspective. |
| 1. Adopt a Secure Coding Standard | Utilizing a specific standard can help to implement a secure system more easily. This can also help companies comply with certain regulatory requirements and increase consumer trust. |

### 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.

All code examples below are from the SEI CERT C++ Coding Standard unless otherwise noted.

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Do not use decimal values as loop counters |

| **Noncompliant Code** |
| --- |
| Decimal values cannot always be represented perfectly in binary and this function may loop 9 or 10 times, depending on how it is implemented. This can result in undesired behavior. |
| void func(void) {    for (float x = 0.1f; x <= 1.0f; x += 0.1f) {      /\* Loop may iterate 9 or 10 times \*/    }  } |

| **Compliant Code** |
| --- |
| This function uses an integer for the loop counter and uses that integer to derive the floating point value. |
| #include <stddef.h>    void func(void) {    for (size\_t count = 1; count <= 10; ++count) {      float x = count / 10.0f;      /\* Loop iterates exactly 10 times \*/    }  } |

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

| **Principles(s): 4-** Keep it simple: Using a floating point value for a loop counter only adds to the complexity and doesn’t work correctly anyways. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | For-loop-float | Fully checked |
| Clang | 3.9 | Cert-flp30-c | Checked by clang-tidy |
| Parasoft C/C++test | 2022.1 | CERT\_C-FLP30-a | Do not use floating point variables as loop counters |
| Coverity | 2017.07 | MISRA C 2004 Rule 13.4  MISRA C 2012 Rule 14.1 | Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not perform operations on unsigned integers that will result in wrap-around |

| **Noncompliant Code** |
| --- |
| If the sum of the two integers passed to this function exceed the maximum size for an integer, the value will wrap. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {    unsigned int usum = ui\_a + ui\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This function does a precondition check to see if the operation will wrap. |
| #include <limits.h>    void func(unsigned int ui\_a, unsigned int ui\_b) {    unsigned int usum;    if (UINT\_MAX - ui\_a < ui\_b) {      /\* Handle error \*/    } else {      usum = ui\_a + ui\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** 10 - Adopt a Secure Coding Standard: Utilizing secure coding practices would prevent wrap-around operations to begin with. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | Integer-overflow | Fully Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT30 | Implemented |
| Polyspace Bug Finder | R2022b | CERT C:Rule INT30-C | Checks for unsigned integer overflow and unsigned integer constant overflow. |
| TrustInSoft Analyzer | 1.38 | Unsinged overflow | Exhaustively verified |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Ensure there is enough space allocated for a string to include the text and a null terminator |

| **Noncompliant Code** |
| --- |
| This code does not take the null terminator into account and will attempt to write past the end of the destination array. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {     size\_t i;       for (i = 0; src[i] && (i < n); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| This loop stops before the last element of the array, leaving room for the null terminator. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {     size\_t i;       for (i = 0; src[i] && (i < n - 1); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

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

| **Principles(s):** 10 - Adopt a Secure Coding Standard: Utilizing secure coding practices would make sure that such range issues would not occur to begin with. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Exhaustively verified |
| CodeSonar | 7.1p0 | LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\* | Buffer overrun, type overrun, no space for null terminator |
| Astree | 22.04 | Intrinsic | Reports all buffer overflows |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Parameterize queries to prevent injection attacks |

| **Noncompliant Code** |
| --- |
| Adding additional code in the email field could allow for executing additional SQL commands |
| $name = $\_REQUEST['name'];  $email = $\_REQUEST['email'];  $sql = "INSERT INTO CustomerTable (Name, Email)  VALUES ('$name', '$email')"; |

| **Compliant Code** |
| --- |
| This code uses parameters to construct the query, preventing additional operations from being performed. |
| $name = $\_REQUEST['name'];  $email = $\_REQUEST['email'];  $params = array($name, $email);  $sql = 'INSERT INTO CustomerTable (Name, Email) VALUES (?, ?)'; |

**Code examples from https://techcommunity.microsoft.com/t5/sql-server-blog/how-and-why-to-use-parameterized-queries/ba-p/383483**

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

| **Principles(s):** 1 - Validate Input Data: User data should be validated to make sure nefarious activity, like SQL injection, cannot occur.  7 - Sanitize Data Sent to Other Systems: Similarly, we need to make sure data sent to the database does not contain unexpected queries. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Low | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Sqlmap | 1.3.4 | Intrinsic | Automates checking for SQL injection flaws |
| Havij | 1.16 | Intrinsic | GUI tool for automated SQL injection detection |
| Burp | 2022.11.3 | Web vulnerability scanner | Finds web application vulnerabilities like SQL injection |
| Acunetix | [Insert text.] | Intrinsic | Automated tool to check for web application vulnerabilities like SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Do not use memory locations that have been freed |

| **Noncompliant Code** |
| --- |
| This code attempts to use the pointer ‘s’ after it has been deallocated from memory. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| This code does not deallocate the pointer until after the operations that use it. |
| #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):** 2 - Heed Compiler Warnings: Many compliers will give errors when trying to access a memory location whose pointer has been deallocated. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.04 | Dangling\_pointer\_user | Reports all access to freed allocated memory |
| CodeSonar | 7.1p0 | ALLOC.UAF | Use after free check |
| Parasoft C/C++ test | 2022.1 | CERT\_C-MEM30-a | Do not use resources that have been freed |
| Polyspace Bug Finder | R2022b | CERT C: Rule MEM30-C | Checks for accessing previously freed pointers and freeing the same. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Ensure assertions do not have side-effects |

| **Noncompliant Code** |
| --- |
| The aFunction() method could effect other parts of the code and if assertions are removed for production builds, this could have unintended consequences. |
| ASSERT (aFunction(1) != 0) |

| **Compliant Code** |
| --- |
| Using the VERIFY macro will evaluate the expression passed to it, but will not check the result in the production build. |
| VERIFY (aFunction(1) != 0) |

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

| **Principles(s): 9 -**  Use effective Quality Assurance Techniques: A static analysis of code should be able to easily identify cases where assertions could have side effects. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 16.0 | Bugprone-assert-side-effect | Finds assertions with side effects |
| CodeQL | 2.11.4 | side-effect-in-assert | Finds assertions with side effects |
| Cppcheck | 2.10.99 | Checkassert.cpp | Finds assertions with side effects |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle every exception |

| **Noncompliant Code** |
| --- |
| The f() and main() functions in this code do not provide a way to handle exceptions. The program will terminate. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| This code has a try/catch block in the main method that will catch any exceptions for methods that it calls, or for methods those methods call. |
| 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):** 2 - Heed Compiler Warnings: Many compliers will warn when exceptions are unhandled.  Use Effective Quality Assurance Techniques: Automated testing should also be able to identify unhandled exceptions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.1p0 | LANG.STRUCT.UCTCH | Finds unreachable catch statements |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-ERR51-a CERT\_CPP-ERR51-b | Makes sure code always catches exceptions |
| Astree | 22.10 | main-function-catch-all early-catch-all | Partially checked |
| Rule Checker | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data Values | STD-008-CPP | Do not use a value that is out of range to cast an enumerated type |

| **Noncompliant Code** |
| --- |
| This code casts enumVar before checking if intVar is in range. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This code checks the bounds of the enumerated type before casting a value with it. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** 9 - Use Effective Quality Assurance Techniques: Static analysis tools should be able to detect potential range errors such as this. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Cast-integer-to-enum | Partially checked |
| CodeSonar | 7.1p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| RuleChecker | 22.10 | Cast-integer-to-enum | Partially checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| String Correctness | STD-009-CPP | Ensure an element is within the range of a string |

| **Noncompliant Code** |
| --- |
| This code capitalizes the first letter of a string, but does not check if the string is empty, which would result in undefined behavior. |
| #include <string>  #include <locale>    void capitalize(std::string &s) {    std::locale loc;    s.front() = std::use\_facet<std::ctype<char>>(loc).toupper(s.front());  } |

| **Compliant Code** |
| --- |
| This code adds a statement to check for an empty string before it attempts to access the first element. |
| #include <string>  #include <locale>    void capitalize(std::string &s) {    if (s.empty()) {      return;    }      std::locale loc;    s.front() = std::use\_facet<std::ctype<char>>(loc).toupper(s.front());  } |

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

| **Principles(s):** 1 - Validate User Input: The program should make sure the string is not empty before calling operations on its first element. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Assert\_failure | Fully implemented |
| CodeSonar | 7.1p0 | LANG.MEM.BO LANG.MEM.BU LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU | Checks for buffer over/underrun, tainted buffer access, type over/underrun |
| Parasoft C/C++ test | 2022.1 | |  |  | | --- | --- | |  | CERT\_CPP-STR53-a | |  |  | | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2022b | CERT C++: STR53-CPP | Checks for out of bounds access, tainted index access, dereferenced pointer access |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Random Numbers | STD-010-CPP | Do not use rand() when strong random numbers are needed |

| **Noncompliant Code** |
| --- |
| This code appends a random number to form an ID number. However, the numbers this function produces can be predictable. |
| #include <cstdlib>  #include <string>    void f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    id += std::to\_string(std::rand() % 10000);    // ...  } |

| **Compliant Code** |
| --- |
| This code uses a stronger random number generation algorithm and will be closer to truly random numbers. |
| #include <random>  #include <string>    void f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    std::uniform\_int\_distribution<int> distribution(0, 10000);    std::random\_device rd;    std::mt19937 engine(rd());    id += std::to\_string(distribution(engine));    // ...  } |

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

| **Principles(s):** 10 - Adopt a Secure Coding Standard: Standards referring to the generation of secure random numbers could ensure that they have enough randomness to suite the application. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | |  |  | | --- | --- | |  | bad-function (AUTOSAR.26.5.1A) | | |  |  | | --- | --- | |  | Fully checked | |
| Clang | 4.0 | Cert-msc50-cpp | Checked by clang-tidy |
| Éclair | 1.2 | CC2.MSC30 | Fully Implemented |
| Parasoft C/C++ test | 2022.1 | CERT\_CPP-MSC50-a | Do not use the rand() function for generating pseudorandom numbers |

### 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.

The current DevOps processes are a good starting point but has room for improvement as far as integrating security. During the design phase, IDEs could have security plug-ins added to analyze code while it is being developed. Additional testing could also be added during the pre-production phase, such as fuzz testing and “chaos” testing. This would ensure a more resilient product. After release, penetration testing could be performed on the production environment. Careful attention should also be paid to the security aspect of the organization’s technical debt.

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

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | This refers to data that is not currently being used. For example, passwords stored on a hard drive of a server. Any sensitive data should always be encrypted using NIST approved algorithms, such as SHA-256 to ensure that information cannot be decrypted. |
| Encryption at flight | This refers to data currently in transit on a network. This data needs to be encrypted so that network packets that are intercepted cannot be utilized. When being sent over the web, this can be done by using the secure sockets layer (SSL), combined with transport layer security (TLS). Internal network traffic can be secured using encryption at the application level. |
| Encryption in use | This refers to encrypting data while it is in use by a program. This ensures data is never left unencrypted. Without this, sensitive data can be resting in memory as plaintext. This can be prevented by using secure memory locations or homomorphic encryption, which allows operations to be performed on encrypted data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to controlling who can access a system, typically in the form of a username/password login. Two factor authentication can also be implemented to add an additional layer of security, requiring the user to enter a code sent to a phone number or email, for example. Every system should be secured this way at some level so that unauthorized third parties cannot use it. |
| Authorization | Authorization refers to controlling what resources a user can access. Typically, the user should only be granted just enough permissions to perform their given function. By default, resources should be denied to everyone except those explicitly granted permission. Using standardized roles can make it easier to control permissions when adding new users by granting access based on, for example, job title. |
| Accounting | Accounting refers to tracking activity in a system. This can come in the form of monitoring who performs a certain action on a given resource. This can make it easier to detect unauthorized access, as well as for troubleshooting. |

**\***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 | 11/13/2022 | Initial Draft | Michael Shunk |  |
| 1.2 | 12/4/2022 | Final Draft | Michael Shunk |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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