**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 | Input validation is key to avoiding unexpected behavior in functions and processes that require input from outside sources. Input that is out of the scope of the application can cause vulnerabilities such as SQL injection. Using preset inputs that are triggered by an API can provide a layer of defense that is needed especially when handling database queries. |
| 1. Heed Compiler Warnings | Compiler warnings can range from informational to critical. It is a common mistake for developers to ignore them since they do not halt compiling and running a project. Reviewing the warnings and making the necessary changes can reduce the risk of potential vulnerabilities in outdated libraries or poorly written code. |
| 1. Architect and Design for Security Policies | Unfortunately, in many cases security is often an afterthought in the development of a project. Designing the architecture of a system with security in mind can produce a cohesive layered system that protects from many potential attacks. Implementing security as the last step can require code rewrites or poorly implemented security practices that compromise the system. |
| 1. Keep It Simple | Simplicity is important not only in security but development as a whole. Many believe that complex applications that maximize efficiency are preferred over readability. In an enterprise environment this is not the case. Many others of varying degrees of knowledge will need to modify and test the code base. If it is not easy to comprehend this can delay work or produce subpar work. |
| 1. Default Deny | This principle sets the precedence that if access hasn’t been explicitly approved then the request should be denied. Each system should have rules on who is allowed, and all others will be rejected. This prevents forgotten edge cases that can gain access by mistake. |
| 1. Adhere to the Principle of Least Privilege | Access rules and privileges is important in any enterprise even outside of software. People involved in the business should have varying degrees of access to information depending on the relevance to their job. By adhering to the principle of least privilege, everyone at the company is given access to the minimum amount of resources required to do their job. This helps prevent unwanted or accidental use of company assets and resources. |
| 1. Sanitize Data Sent to Other Systems | Data sanitization is the process of data scrubbing to prevent release of sensitive information when transferring data and reusing or throwing away hardware. If data needs to be sent to another system or third party it is important to review the data to make sure no unnecessary information is being sent. Using techniques such as masking to actually remove the data from the hardware prevents attackers from reading the contents of deleted information. |
| 1. Practice Defense in Depth | Defense in depth is the practice of layering security measures to prevent unwanted access or tampering of systems. By implementing various levels of security at different points of the application it makes it more difficult for hackers to cause harm. To implement this principle correctly it must be thought in the beginning of a project when designing the architecture. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance is an important part of any software development lifecycle. Using various techniques to ensure quality and secure coding practices is important to improving security. Unit, system and other forms of testing are necessary to finding flaws in an application and fixing them before they can be exploited. |
| 1. Adopt a Secure Coding Standard | Secure coding standards are important to setting the standards the company will hold themselves to when implementing securing policies in the organization. The policies and standards should be exhaustive and complete so that members can be held accountable to working by them. This helps create a work culture that incorporate security into its day-to-day practice. |

### 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** | [**Ensure that operations on signed integers do not result in overflow**](https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow) |
| --- | --- | --- |
| **Data Value** | STD-001-Cpp | Integer overflow can produce unexpected behavior including access to unauthorized sections of memory. |

| **Noncompliant Code** |
| --- |
| Addition arithmetic operation resulting in integer overflow |
| int i;  i = 2^32 + 1; |

| **Compliant Code** |
| --- |
| Subtraction arithmetic operation without overflow occurring |
| int i;  i = 4 – 2; |

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

| **Principles(s):** Use Effective Quality Assurance Techniques: Error handling is necessary to ensure that integer overflow does not occur. Custom exceptions can be used to detect and halt overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **integer-overflow** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **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 |

#### Coding Standard 2

| **Coding Standard** | **Label** | [**Ensure that integer conversions do not result in lost or misinterpreted data**](https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data) |
| --- | --- | --- |
| **Data Type** | STD-002-CPP | Data type conversions must be guaranteed not to lose or alter the source information. Sources that are untrusted may not be validated in the same ways as done internally. |

| **Noncompliant Code** |
| --- |
| Numerical value is provided as a string instead of an integer. |
| 4 = “2” + 2; |

| **Compliant Code** |
| --- |
| Numerical provided value is an integer as expected |
| 4 = 2 + 2; |

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

| **Principles(s):** Validate Input Data: Data from third party services may be corrupt or unformatted. This can lead to undefined behavior in the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **signed\_downcast** | Exhaustively verified. |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | N/A | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Ensure there is enough space for character data and null terminator in strings** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Strings are character arrays that are ended with a null terminator value. If the space allotted does not equal the number of characters and the null terminator then a buffer overflow will occur, and the string will no longer hold the correct value. |

| **Noncompliant Code** |
| --- |
| Cin is used to take user input of 20 characters which exceeds the array allotment. |
| char array[5];  std::cin >> array; |

| **Compliant Code** |
| --- |
| Cin is used to take user input of 2 character which is within the array allotment. |
| char array[5];  std::cin >> array; |

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

| **Principles(s):** Adopt a Secure Coding Standard: The application should determine there is not enough space for the data and throw an error without writing to memory. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | N/A | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| [Coverity](https://www.securecoding.cert.org/confluence/display/seccode/Coverity) | 2017.07 | **STRING\_OVERFLOW**  **BUFFER\_SIZE**  **OVERRUN**  **STRING\_SIZE** | Fully implemented |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **mem\_access** | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/144ae03a)). |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Ensure that queries are created as set parameters.** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | By creating a data access layer to a system, services can be used to perform set actions rather than pass user input as queries to the database. This will help prevent SQL injection. |

| **Noncompliant Code** |
| --- |
| Query is taken from user input. |
| std::cin >> query;  void function (String query) {  db.query(query);  } |

| **Compliant Code** |
| --- |
| Predefined query is chosen through user input. |
| std::cin >> query;  switch(query){  case: 1  db.query(“SELECT \* FROM table\_name;”);  default:  std::cout << “No query selected” << std::endl;  } |

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

| **Principles(s): Practice Defense in Depth: Provide layers between the database and client to prevent direct access from users.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/java/Coverity) | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Findbugs](https://wiki.sei.cmu.edu/confluence/display/java/Findbugs) | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| [Fortify](https://wiki.sei.cmu.edu/confluence/display/java/Fortify) | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Perform garbage collection techniques to avoid stack overflow.** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | C++ does not automatically perform garbage collection and objects must be manually destroyed to free up space in memory. |

| **Noncompliant Code** |
| --- |
| Objects are created in while loop that never ends and are not destroyed |
| char q = ‘n’;  while (q == ‘n’) {  Object object;  } |

| **Compliant Code** |
| --- |
| Objects are create and destroyed in while loop that never ends. |
| char q = ‘n’;  while (q == ‘n’) {  Object object;  delete object;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: The application should be programmed to assume that a stack overflow may occur and allow the application to terminate safely. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | N/A | N/A | Runtime detection |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **invalid\_dynamic\_memory\_allocation dangling\_pointer\_use** | N/A |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Utilize assertions to verify pre-conditions of an application** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Applications should utilize assertions to verify the state of the application was initialized correctly. If it was started with problems that compromise the system, then the application should be terminated. |

| **Noncompliant Code** |
| --- |
| System should initialize with int a = 1 but is incorrectly started with a = 2. |
| int a = 2;  assert(a == 1); |

| **Compliant Code** |
| --- |
| System is initialized with int a = 1. |
| int a = 1;  assert(a == 1); |

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

| **Principles(s):** Use Effective Quality Assurance Techniques: Diagnostics should be ran to ensure that the system runs in a secure state using assertions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **LANG.FUNCS.ASSERTS** | Not enough assertions |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ASSERT\_SIDE\_EFFECT** | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2021.2 | **CERT\_C-MSC11-a** | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Detect and handle errors that can be thrown by third party libraries** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Libraries should be documented with the different types of errors that can be thrown when using them. These should be caught and handled accordingly to avoid undefined behavior. |

| **Noncompliant Code** |
| --- |
| Function is used that inserts input into an array at the specified spot. The array in this example holds 5 spaces and throws an exception that the entry is out-of-bounds which is not caught. |
| examplefunct(10, “entry”); |

| **Compliant Code** |
| --- |
| Function is used that inserts input into an array at the specified spot. The array in this example holds 5 spaces and throws an exception that the entry is out-of-bounds which is caught. |
| try{  examplefunct(10, “entry”);  }  catch (exception& e){  std::cout << e.what() << std::endl;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard: Ensure that third party library exceptions are caught and handled appropriately. This allows the applications to terminate or handle errors correctly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **MISRA C 2012 Rule 22.8**  **MISRA C 2012 Rule 22.9**  **MISRA C 2012 Rule 22.10** | Implemented |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **pointer arithmetic** | Exhaustively verified. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not attempt to access freed memory** |
| --- | --- | --- |
| Memory Protection | STD-008-CPP | Memory that has been marked as free to delete will be undefined if accessed by the program. |

| **Noncompliant Code** |
| --- |
| A pointer is assigned to a node which is marked as free to delete. The pointer is then accessed again in the next statement. |
| node \*p = head;  free(p);  std::cout << \*p << std::endl; |

| **Compliant Code** |
| --- |
| A pointer is assigned to a node and before it is marked to be free to delete all of the operations needed are performed on it. |
| node \*p = head;  std::cout << \*p << std::endl;  free(p); |

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

| **Principles(s):** Architect and Design for Security Policies: Ensure that protections are in place to avoid access to memory that has already been deallocated. Pointers that are no longer relevant should not be used. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **dangling\_pointer\_use** | N/A |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | N/A | N/A | Runtime detection |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not user floating point variables to loop counters** |
| --- | --- | --- |
| Data Type | STD-009-CPP | Floating point values can represent large numbers but are also confined to bit limitations like integers. The value may also be confined to a small number range of whole numbers due to the precision set. |

| **Noncompliant Code** |
| --- |
| In some implementations of binary 0.1 is a repeating fraction that cannot be expressed exactly as floating point. Depending on the architecture this example could loop either 9 or 10 times. |
| void func(void) {    for (float x = 0.1f; x <= 1.0f; x += 0.1f) {    }  } |

| **Compliant Code** |
| --- |
| The example below uses integers instead of floating-point values as a counter. This will loop 10 times regardless of the architecture. |
| void func(void) {    for (int x = 1; x <= 10; x++) {    }  } |

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

| **Principles(s):** Keep It Simple: Counters should be integers rather than floating point in order to avoid complications with bit limitations. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **for-loop-float** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-FLP30** | Fully implemented |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **MISRA C 2004 Rule 13.4**  **MISRA C 2012 Rule 14.1** | Implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Ensure that division operations do not divide by zero** |
| --- | --- | --- |
| Data Value | STD-010-CPP | Dividing by 0 is undefined in arithmetic and will cause abnormal behavior in the program. |

| **Noncompliant Code** |
| --- |
| An integer is divided by 0. |
| int i;  i = 2/ 0; |

| **Compliant Code** |
| --- |
| An integer is not divided by 0. |
| int i;  i = 2/ 2; |

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

| **Principles(s):** Adopt a Secure Coding Standard: Maintain secure coding practices with mathematical operations to avoid unnecessary errors. Values should not be divided by zero to ensure no undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **int-division-by-zero**  **int-modulo-by-zero** | Fully checked |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **DIVIDE\_BY\_ZERO** | Fully implemented |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Starting at an entrance point in assess and plan, leaders and designers must determine the requirements of the client and transform it into an actionable plan. Security can be implemented here by having experts sit in the meeting to determine security requirements based on the system design or regulatory constraints of the field. Static tests can be used to determine appropriate versions and frameworks are used to limit vulnerabilities. Building can then take place using secure coding practices that developers have been trained to perform. QA can then run unit tests to check code functionality along with common security vulnerabilities such as overflows or SQL injection. Once the code has been pushed to production, logging and monitoring must be implemented to ensure constant security throughout the lifecycle. At this point authentication, authorization, and accounting are used to improve security layers outside of the application. Any breaches or vulnerabilities found will be consulted with the security team to detect, isolate, and control. Recommendations will be sent to the project leads who can then start the pre-production lifecycle over again for a new feature or fix.

### 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 | Likely | Low | High | 2 |
| STD-002-CPP | High | Likely | Medium | High | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Probable | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | High | Likely | Medium | High | 1 |
| STD-008-CPP | High | Likely | Medium | High | 1 |
| STD-009-CPP | Low | Likely | Low | Low | 2 |
| STD-010-CPP | Low | Likely | Medium | 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 | All sensitive data at rest must have some form of encryption. Encryption at rest is the act of encrypting data that is being stored while not in use. The purpose of this is to create a layer of defense that the attacker must breach to read the data if it is obtained after previous layers have been defeated. Various encryption methods are available depending on the use of the data. |
| Encryption at flight | When transmitting data, a VPN must be utilized. Encryption at flight is encrypting data that is being transmitted. This is often used for information sent over the internet or between two devices. If the malicious attacker intercepts the data, then they will not be able to access it without the key. VPNs are a common way to hide the source of the day and provide encryption. |
| Encryption in use | Credit cards numbers must be encrypted while in use. Encryption in use ensures that sensitive data is never left unsecured. The programs that utilize this method can operate on the data in the encrypted state which leaves zero points of contact that a user could intercept and obtain unencrypted data. Highly sensitive data that does not need to be read by humans could utilize this method to add an additional layer of security. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Google 2FA must be utilized for all system logins. Authentication is the process of verifying that the user accessing the system is who they say they are. User logins and multi-factor authentication allow confirmation and secure user access. |
| Authorization | Every job role must have set permissions determined based on the requirements of the position. Once a user has been confirmed, authorization is the concept of applying levels of access to different user roles. Users by default are given privilege to the least amount of access necessary and are allowed additional functionality if required in the future. |
| Accounting | Logging on changes done to user accounts by associates must be logged and stored for a minimum of 7 years. Accounting allows for logging and tracking of users or processes in a system. Being able to track changes can help security experts determine breaches or vulnerabilities in the system. |

**\***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 |  |
| 2.0 | 01/13/22 | Milestone One | Josh Ryther | Trevor Hodde |
| 3.0 | 2/5/22 | Project One | Josh Ryther | Trevor Hodde |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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