**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 | Inputs should be validated to ensure that they meet the correct format, type , and range. This is used to prevent vulnerabilities from injection attacks, buffer overflows, or system crashes from improper input. Limiting the input limits that user’s ability to misuse or abuse the system and its data. |
| 1. Heed Compiler Warnings | Warnings from compilers occur to inform the developer of potential issues within the code. These can point to vulnerabilities such as type mismatches or depreciated functions. These should be addressed to reduce security risks or risks of error within the software. |
| 1. Architect and Design for Security Policies | Security should be built into the architecture of the software from the start of the designing process. This allows for clear security requirements to be implemented and for each software addition to be compared against the requirements. |
| 1. Keep It Simple | Keeping a simple design allows for code to be easier to understand, maintain, and audit. This reduction in complexity helps prevent hidden bugs from existing through complex interactions and designs. It makes auditing easier, so these bugs can be found quicker and repaired easier. |
| 1. Default Deny | Systems should default to denying access. This means that access should only be permitted when it is a necessity. This limits the systems that vulnerabilities can interact with, helping to maintain security. |
| 1. Adhere to the Principle of Least Privilege | Every user and component should operate with minimum privileges necessary to complete their task. This lowers the chance of a compromise spreading through the system and helps contain treats. |
| 1. Sanitize Data Sent to Other Systems | Data that is moving from one system to another must be sanitized. This involves stripping data of characters that could be interpreted as code. This prevents SQL injections and adds security to the systems before invoking the data. |
| 1. Practice Defense in Depth | Multiple layers of defense are needed to create overlap and ensure coverage across all layers. The importance of this is to ensure that the multiple layers of defense do not hinder the functioning of the system but provide full coverage against vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance can be used to prevent flaws in coding. Quality assurance practices allow for vulnerabilities to be discovered early in development. This can be used to make corrections before system interactions make it more difficult to fix. It also helps to prevent future vulnerabilities of similar kinds because of them being found in other places. |
| 1. Adopt a Secure Coding Standard | This coding standard allows for consistent framework to be used to maintain security. It can be used to avoid known bad practices and encourage secure practices. This also assists with maintainability and understandability. |

### 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 create incompatible declarations of the same object |

| **Noncompliant Code** |
| --- |
| The variable x, is defined with different types in different scopes. This can cause compiler confusion or logic bugs from the developers. The developer could want to use the global definition, but is using the local variable instead. |
| int x = 12;  void copy() {  float x = 134.15; //this is used within the function over the global x  return x; // would return 134.15 local value  } |

| **Compliant Code** |
| --- |
| The variable x, is defined with different types in different scopes. This can cause compiler confusion or logic bugs from the developers. The developer could want to use the global definition, but is using the local variable instead. |
| int x = 12;  void copy() {  float y = 134.15; //this can be used or x can be used  return y; // would return 12 global value  } |

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

| **Principles(s):** (10) Adopt a Secure Coding Standard – This ensures that variable declarations are consistent and avoids errors from incomplete redeclarations. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium – Detectable/Not repairable | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | **type-compatibility**  **type-compatibility-link**  **distinct-extern** | Fully Checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-DCL40** | Fully Implemented |
| Cppcheck Premium | 24.11.0 | **premium-cert-dcl40-c** | Checks for redeclarations and variable shadowing |
| Polyspace Bug Finder | R2024b | [CERT C: Rule DCL40-C](https://www.mathworks.com/help/bugfinder/ref/certcruledcl40c.html) | Checks for declaration mismatch |

#### Coding Standard 2

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

| **Noncompliant Code** |
| --- |
| The variable x is defined but not initialized with a value. This can give undefined behaviors. |
| int x; // no initialized value  std::cout << x; // printing an undefined value can cause errors |

| **Compliant Code** |
| --- |
| The correct way to handle this situation would be to ensure that a value is initialized before calling it. |
| int x = 12;  std::cout << x; // variable is initialized to 12 so it will print |

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

| **Principles(s):** (9) Use Effective Quality Assurance Techniques – Uninitialized values can lead to undefined behaviors and bugs. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium – Repairable/Not detectable | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | **uninitialized-local-read**  **uninitialized-variable-use** | Fully Checked |
| CodeSonar | 9.0p0 | **LANG.MEM.UVAR** | Uninitialized variable |
| Cppcheck Premium | 24.11.0 | **uninitvar uninitdata uninitstring uninitMemberVar uninitStructMember** | Checks uninitialized values(var, data, string, member var, and struct member |
| Polyspace Bug Finder | R2024b | [CERT C: Rule EXP33-C](https://www.mathworks.com/help/bugfinder/ref/certcruleexp33c.html) | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for the string has sufficient space or characters and null terminator |

| **Noncompliant Code** |
| --- |
| The string is defined to be the full length of the character limit and does not allow for a null character. |
| num str1[10]; // this allows for 10 characters  strcpy(str, “0123456789”); //this string is 10 characters preventing null terminator |

| **Compliant Code** |
| --- |
| The string was made to be 1 longer than the size of the input to allow the null terminator. It would be better to limit the input to 10 or check that it is 10 or less to ensure that this is being enforced. |
| num str1[11]; // this allows for 11 characters  strcpy(str, “0123456789”); //this string is 10 characters allowing a null terminator |

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

| **Principles(s):** (7) Sanitize Data Sent to Other Systems – This helps to ensure that strings are sized correctly to prevent injections and corruption during system to system transfers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High – Not detectable/Not repairable | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | **stream-input-char-array** | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| CodeSonar | 9.0p0 | **LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\*** | Buffer overrun Type overrun No space for null terminator A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| LDRA tool suite | 9.7.1 | **489 S, 109 D, 66 X, 70 X, 71 X** | Partially implemented |
| Polyspace Bug Finder | R2024b | [CERT C: Rule STR31-C](https://www.mathworks.com/help/bugfinder/ref/certcrulestr31c.html) | Rule partially covered. |

#### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| This takes in raw user input and uses it for a queary. This allows for SQL injection |
| std::string username = getUserInput();  std::string query = “SELECT \* FROM users WHERE name = ‘” + username + “’;”;  querySQL(query); // does not sanatize user input |

| **Compliant Code** |
| --- |
| The compliant version sanitized and used a parameterized queary. This uses SQLite to make a prepared statement |
| sqlite3\_stmt\* stmt;  sqlite3\_prepare\_v2(db, "SELECT \* FROM users WHERE name = ?", -1, &stmt, NULL);  sqlite3\_bind\_text(stmt, 1, username.c\_str(), -1, SQLITE\_STATIC);  sqlite3\_step(stmt);  sqlite3\_finalize(stmt); |

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

| **Principles(s):** (1) ValidateInput Data – This ensures that user input is structured and parameterized to prevent injection attacks |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium – Repairable/Not detectable | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  | Supported, but no explicit checker |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-MSC32** |  |
| Cppcheck Premium | 24.11.0 | premium-cert-msc32-c |  |
| PC-lint Plus | 1.4 | 2460, 2461, 2760 | Fully supported |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Allocate sufficient memory for an object. Failing to do this can lead to buffer overflows and crashes. |

| **Noncompliant Code** |
| --- |
| The code allocates memory for an int using sizeof(ptr) which can lead to under-allocation of memory |
| int\* ptr = (int\*)malloc(sizeof(ptr)); //allocates size of pointer instead of int  \*ptr = 58; |

| **Compliant Code** |
| --- |
| This version allocates the correct amount of memory to the int and not the pointer |
| int\* ptr = (int\*)malloc(sizeof(\*ptr)); // slight difference that allocates size of int  \*ptr = 58; |

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

| **Principles(s):** (3) Architect and Design for Security Policies – This requires developers to ensure that memory is properly allocated as part of a secure software design |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High-Not detectable/Not Repairable | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | **malloc-size-insufficient** | Partially checked  Besides direct rule violations, all undefined behaviour resulting from invalid memory accesses is reported by Astrée. |
| CodeSonar | 9.0p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW ALLOC.SIZE.TRUNC IO.TAINT.SIZE MISC.MEM.SIZE.BAD LANG.MEM.BO LANG.MEM.BU LANG.STRUCT.PARITH LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU** | Addition overflow of allocation size Addition overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Truncation of allocation size Tainted allocation size Unreasonable size argument Buffer Overrun Buffer Underrun Pointer Arithmetic Pointer Before Beginning of Object Pointer Past End of Object Tainted Buffer Access Type Overrun Type Underrun |
| Cppcheck Premium | 24.11.0 | **premium-cert-mem35-c** |  |
| Polyspace Bug Finder | R2024b | [CERT C: Rule MEM35-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemem35c.html) | Checks for:   * Pointer access out of bounds * Memory allocation with tainted size   Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Do not use input data for assertions because assertions can be disabled for production |

| **Noncompliant Code** |
| --- |
| The function uses an external input to test an assertion |
| void test(int userInput) {  assert(userInput < 0); // this relies on input from user  //test function  } |

| **Compliant Code** |
| --- |
| This version removes the assertion and uses an if statement to complete the test instead. Disabling assertions would have no affect on this |
| void test(int userInput) {  if (userInput < 0){ // this tests without assertion  //test function  }  } |

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

| **Principles(s):** (2) Heed Compiler Warnigns – Compilers will warn against this, the assertions may be optimized out during runtime causing insecure behavior |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | High – Not detectable/Not repairable | low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | **pointer-qualifier-cast-const pointer-qualifier-cast-const-implicit** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-EXP05** | Fully implemented |
| CodeSonar | 9.0p0 | **LANG.CAST.PC.CRCQ** | Cast removes const qualifier |
| Polyspace Bug Finder | R2024b | [CERT C: Rec. EXP05-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.exp05c.html) | Checks for cast to pointer that removes const qualification (rec. fully supported) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not use setjmp or longjmp. These can bypass normal stack unwinding, which can leak resources. |

| **Noncompliant Code** |
| --- |
| This code uses longjmp for error handling which can cause resource leaks |
| std::longjmp(env, 1); // skips deconstructor, leaving resources open |

| **Compliant Code** |
| --- |
| [Compliant description] |
| throw std::runtime\_error("Error needs handled"); // Uses exception instead of jmp |

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

| **Principles(s):** (8) Practice Defense in Depth – This produces structure exception handling that allows for efficient resource management |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **include-setjmp** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-ERR52** |  |
| CodeSonar | 9.0p0 | **BADFUNC.LONGJMP BADFUNC.SETJMP LANG.PREPROC.INCL.SETJMP\_H** | Use of longjmp Use of setjmp Use of <setjmp.h> |
| Polyspace Bug Finder | R2024b | [CERT C++: ERR52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr52cpp.html) | Checks for use of setjmp/longjmp (rule fully covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Data value | [STD-008  -CPP] | Use volatile for variables modified in signal handlers. If the variable is in both the signal handler and main program without being volatile, the compiler may optimize out necessary reads. |

| **Noncompliant Code** |
| --- |
| This modifies the variable in both the main program and the flagHandler, making it possible to be optimized out. |
| int sigFlag = 0;  void flagHandle(int signum) {  sigFlag = 1; // non volatile  } |

| **Compliant Code** |
| --- |
|  |
| volatile sig\_atomic\_t sigFlag = 0;  void handler(int signum) {  sigFlag = 1; // sigflag is defined as volatile making this safe  } |

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

| **Principles(s):** (1) Validate Input Data - This promotes the safe handling of signal input s without compiler optimization issues |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High – Not detectable/Not repairable | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2025.1 | **C++2888** |  |
| Klocwork | 2025.1 | **CERT.MSC.SIG\_HANDLER.POF** |  |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-MSC54-a** | Properly define signal handlers |
| Polyspace Bug Finder | R2024b | [CERT C++: MSC54-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc54cpp.html) | Checks for unsafe signal handlers (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory Protection | [STD-009-CPP] | Do no access invalid memory. This can cause out of bounds pointer leading to crashes and vulnerabilities. |

| **Noncompliant Code** |
| --- |
| The pointer is defined a null and than defined as being 5. This dereferences the null pointer causing invalid memory access. |
| int\* p = nullptr;  \*p = 5; // this is dereferencing a null |

| **Compliant Code** |
| --- |
| The value is initializes, the address is referenced, and then the memory is modified. |
| int num = 10;  int\* p = &num;  \*p = 1; // valid referencing |

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

| **Principles(s):** (8) Practice Defense in Depth – This ensures that memory is only accessed when it is valid to avoid failure from null pointer dereferences |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **dangling\_pointer\_use** |  |
| CodeSonar | 9.0p0 | **ALLOC.UAF** | Use after free |
| LDRA tool suite | 9.7.1 | **483 S, 484 S** | Partially implemented |
| Polyspace Bug Finder | R2024b | [CERT C++: MEM50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem50cpp.html) | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | [STD-010-CPP] | Detect error conditions from function calls. If an error is not checked, it can lead to unexpected behaviors. |

| **Noncompliant Code** |
| --- |
| this code calls to open a file without determining if an error occurs. This can lead to unexpected behavior in later code trying to interact with the file. |
| FILE\* readIn = fopen("fileRead.txt", "r"); // no error checking |

| **Compliant Code** |
| --- |
| This is the code with the addition of an if statement that determines if the file is open. This can be used to exit the function before it tries to interact with the file that had an error opening. |
| FILE\* readIn = fopen("fileRead.txt", "r");  if (!readIn) {  perror("Open failed"); // determines if file is open for future use  } |

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

| **Principles(s):** (9) Use Effective Quality Assurance Techniques – this requires proper error handling to function calls. This catches bugs earlier in development. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | low- Detectable/repairable | medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-ERR34** |  |
| CodeSonar | 9.0p0 | **BADFUNC.ATOF BADFUNC.ATOI BADFUNC.ATOL BADFUNC.ATOLL** | Use of atof Use of atoi Use of atol Use of atoll  Users can add custom checks for uses of other undesirable conversion functions. |
| LDRA tool suite | 9.7.1 | **44 S** | Fully implemented |
| Parasoft C/C++test | 2024.2 | **CERT\_C-ERR34-a** | The 'atof', 'atoi', 'atol' and 'atoll' functions from the 'stdlib.h' or 'cstdlib' library should not be used |

### 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.

In the pre-production portion of the lifecycle, automation can be added into the design portion. Plugins can be utilized to scan for known vulnerabilities. These tools will help ensure that secure coding checklist(OWASP) are met. In the build portion, Static analysis tools(Cppcheck) can be used to automatically flag noncompliant builds. During the verify and test portion, dynamic testing will be triggered on staging requests to verify security before production.

In the production portion, automated monitoring tools will be utilized to ensure monitor for suspicious activity. These tools will log the suspicious activity and be able to be used to determine the cause. During the response part, tools will be used to automate rollback and will assist in response automation. Regular audits will be used to ensure that the code conforms to the security requirements that are laid out. Applying these security automations to different parts of the development will help to prevent vulnerabilities from being abused. It will ensure that constant monitoring and improvements are sought and applied to the software.

### Summary of Risk Assessments

Consolidate 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 | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Probable | Medium | High | 1 |
| STD-003-CPP | High | Likely | High | Medium | 2 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Probable | High | Medium | 2 |
| STD-006-CPP | Medium | Probable | High | Low | 3 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | High | Probable | High | Medium | 2 |
| STD-009-CPP | High | Likely | Medium | High | 1 |
| 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 at rest | Encryption at rest is the protection of stored data on physical locations. This is used to protect data from unauthorized access even if the physical storage is compromised. This would need to be applied to any potentially sensitive data that is stored in physical locations , such as a database or a backup. |
| Encryption in flight | Encryption at flight is the protection of moving data. This would be any data that is traveling over a network. Encryption of this data is used to ensure that man-in-the-middle attacks can’t find sensitive data. It prevents packets from being grabbed on a network. |
| Encryption in use | Encryption in use refers to protecting data while it is being processed. This means that the data should be isolated and contained to ensure that only minimal rights are provided. This means that data can not be taken even from a compromised system. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying the users identity. This is often done through a password or unique identifier. This is to ensure that the user is who they claim to be and should have access. |
| Authorization | Authorization is the specification of access rights to the user. This is to ensure that the user should have access to the data. This can be enforced through role-based access controls that give the least privileges based on the required job. |
| Accounting | Accounting is the process of keeping track of activity within a system this allows for tracking of users interactions with the system and data within it. It allows for a trail to be created to track back to exploited vulnerabilities. |

**\***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 | 06/01/2025 | Initial Coding Standards | Brandon Hollada | [Insert text.] |
| 1.2 | 06/21/2025 | Threat and Automation | Brandon Hollada | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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