**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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 14](#_Toc52464064)

[Coding Standard 7 16](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 20](#_Toc52464067)

[Coding Standard 10 22](#_Toc52464068)

[Defense-in-Depth Illustration 24](#_Toc52464069)

[Project One 24](#_Toc52464070)

[1. Revise the C/C++ Standards 24](#_Toc52464071)

[2. Risk Assessment 24](#_Toc52464072)

[3. Automated Detection 24](#_Toc52464073)

[4. Automation 24](#_Toc52464074)

[5. Summary of Risk Assessments 25](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 26](#_Toc52464076)

[7. Map the Principles 27](#_Toc52464077)

[Audit Controls and Management 28](#_Toc52464078)

[Enforcement 28](#_Toc52464079)

[Exceptions Process 28](#_Toc52464080)

[Distribution 29](#_Toc52464081)

[Policy Change Control 29](#_Toc52464082)

[Policy Version History 29](#_Toc52464083)

[Appendix A Lookups 29](#_Toc52464084)

[Approved C/C++ Language Acronyms 29](#_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 | A technique used to ensure that data entered into any system, website, or web app is valid and meets specific criteria. Validating makes sure no malicious data gets entered. |
| 1. Heed Compiler Warnings | Compiler warnings are messages produced by a compiler regarding program code fragments to be considered by the developer, as they may contain errors. Heeding would be actively fixing and acknowledging the warnings. |
| 1. Architect and Design for Security Policies | Create software architecture and design the software to enforce security policies, it will focus on privileges. |
| 1. Keep It Simple | Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Avoiding complexity and keeping things straightforward should be the objective. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. By default, access is denied. |
| 1. Adhere to the Principle of Least Privilege | Processes should execute with minimal required privileges needed. Users should only be provided access when required. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data before passing the data to other systems could potentially prevent SQL injection attacks. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies. Defense in Depth is the best as you want multiple layers of redundant security. |
| 1. Use Effective Quality Assurance Techniques | Apply a secure coding standard for your target development language and platform. Test early and often are the best techniques while focusing on agile processes. |
| 1. Adopt a Secure Coding Standard | Apply a secure coding standard for your target development language and platform. |

### 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] | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| Two different translation units define a class of the same name with differing definitions. |
| // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |

| **Compliant Code** |
| --- |
| Use of a header file to introduce the object into both translation units |
| // S.h  **struct** S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):**  3: Architect and Design for Security Policies  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading** | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL60** |  |
| **CertC++-DCL60** | 7.4p0 | **CertC++-DCL60** | Function defined in header file Object defined in header file |
| LDRA tool suite | 9.7.1 | **286 S, 287 S** | **286 S, 287 S** |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not call a function with a mismatched language linkage |

| **Noncompliant Code** |
| --- |
| In this compliant solution, the callback\_func() function is given "java" language linkage to match the language linkage for java\_callback. |
| **extern** "java" **typedef** **void** (\*java\_callback)(**int**);    **extern** **void** call\_java\_fn\_ptr(java\_callback callback);  **void** callback\_func(**int**);    **void** f() {    call\_java\_fn\_ptr(callback\_func);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the callback\_func() function is given "java" language linkage to match the language linkage for java\_callback. |
| **extern** "java" **typedef** **void** (\*java\_callback)(**int**);    **extern** **void** call\_java\_fn\_ptr(java\_callback callback);  **extern** "java" **void** callback\_func(**int**);    **void** f() {    call\_java\_fn\_ptr(callback\_func);  } |

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

| **Principles(s):**  1: Validate Input Data  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3033, C++3038** | [Insert text.] |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork?_gl=1*1u782qy*_ga*MzEzNjIwMjkuMTY5NDc5ODY5MQ..*_ga_87WECW6HCS*MTY5NDc5ODY5MS4xLjEuMTY5NDgwMDYxMS4wLjAuMA..) | 2023.1 | **PORTING.CAST.PTR PORTING.CAST.PTR.FLTPNT PORTING.CAST.PTR.SIZE PORTING.CAST.SIZE MISRA.CAST.PTR.UNRELATED MISRA.CAST.PTR\_TO\_INT** | [Insert text.] |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-EXP56-a** | Do not call a function with a mismatched language linkage |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| A std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist (or some other error occurs).. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):**  2: Heed Compiler Warnings |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | assert\_failure |  |
| Helix QAC | 2023.2 | **DF4770, DF4771, DF4772, DF4773, DF4774** |  |
| Klocwork | 2023.3 | NPD.CHECK.CALL.MIGHT NPD.CHECK.CALL.MUST NPD.CHECK.MIGHT NPD.CHECK.MUST NPD.CONST.CALL NPD.CONST.DEREF NPD.FUNC.CALL.MIGHT NPD.FUNC.CALL.MUST NPD.FUNC.MIGHT NPD.FUNC.MUST NPD.GEN.CALL.MIGHT NPD.GEN.CALL.MUST NPD.GEN.MIGHT NPD.GEN.MUST RNPD.CALL RNPD.DEREF |  |
| Parasoft  C/C++tes | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereference |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| Without precautions, the untrusted data may maliciously alter the query. |
| uName = getRequestString("username");  uPass = getRequestString("userpassword");  sql = “SELECT \* FROM Users WHERE Name = " + uName + " AND Pass = " +  uPass + ” |

| **Compliant Code** |
| --- |
| The primary means of preventing SQL injection are sanitization and validation, which are typically implemented as  parameterized queries and stored procedures. |
| PreparedStatement pStmt = PreparedStatement();  std::cin >> username;  std::cin >> userpassword;  sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;”, username,  userpassword}; |

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

| **Principles(s):**  1: Validate Input Data  7: Sanitize Data Sent to Other Systems  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors (see Chapter 8) |
| CodeSonar | 7.4p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| Coverity | 7.5 | **SQLI** **FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| Findbugs | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, s is dereferenced after it has been deallocated. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
|  |
| In this compliant solution, the dynamically allocated memory is not deallocated until it is no longer required. |
| #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  5: Default Deny  6: Adhere to the Principle of Least Privilege  9: Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | **dangling\_pointer\_use** | Supported  Astrée reports all accesses to freed allocated memory. |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-MEM30** | Detects memory accesses after its deallocation and double memory deallocations |
| CodeSonar | 7.4p0 | **ALLOC.UAF** | Use after free |
| Compass/ROSE |  |  |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| Uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int)  + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| For constant expressions, a preprocessor conditional statement may be used. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) +  sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):**  2: Heed Compiler Warnings  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| |  |  | | --- | --- | | Axivion Bauhaus  Suite |  | | 7.2.0 | **CertC-DCL03** |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.4p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| ÉCLAIR | 1.2 | **CC2.DCL03** | Fully implemented |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| Neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| The main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| **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):**  9: Use Effective Quality Assurance Techniques  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| Helix QAC | 2023.3 | **C++4035, C++4036, C++4037** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite)C | 7.2.0 | **CertC++-ERR51** |  |
| Astrée | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-008-CPP) | Guarantee that library functions do not overflow |

| **Noncompliant Code** |
| --- |
| The std::copy() algorithm provides no inherent bounds checking and can lead to a buffer overflow. I |
| #include <algorithm>  #include <vector>    **void** f(**const** std::vector<**int**> &src) {    std::vector<**int**> dest;    std::copy(src.begin(), src.end(), dest.begin());    // ...  } |

| **Compliant Code** |
| --- |
| This compliant solution enlarges the capacity of the vector prior to the copy operation. |
| #include <algorithm>  #include <vector>  **void** f(**const** std::vector<**int**> &src) {    // Initialize dest with src.size() default-inserted elements    std::vector<**int**> dest(src.size());    std::copy(src.begin(), src.end(), dest.begin());    // ...  } |

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

| **Principles(s):**  4: Keep It Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **invalid\_pointer\_dereference** |  |
| CodeSonar | 7.4p0 | **BADFUNC.BO.\*** **LANG.MEM.BO** **LANG.MEM.TBA** | A collection of warning classes that report uses of library functions prone to internal buffer overflows. Buffer Overrun Tainted Buffer Access |
| Helix QAC | 2023.3 | **DF3526, DF3527, DF3528, DF3529, DF3530, DF3531, DF3532, DF3533, DF3534** |  |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-CTR52-a** | Do not pass empty container iterators to std algorithms as destinations |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| MSC | [STD-009-COO] | Do not return from a function declared |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, if the value 0 is passed, control will flow off the end of the function, resulting in an implicit return and undefined behavior. |
| #include <cstdlib>    [[**noreturn**]] **void** f(**int** i) {  **if** (i > 0)  **throw** "Received positive input";  **else** **if** (i < 0)      std::**exit**(0);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function does not return on any code path |
| #include <cstdlib>    [[**noreturn**]] **void** f(**int** i) {  **if** (i > 0)  **throw** "Received positive input";    std::**exit**(0);  } |

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

| **Principles(s):**  3: Architect and Design for Security Policies  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | **invalid-noreturn** | **LANG.STRUCT.RFNR** |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-MSC53** |  |
| Clang | 3.9 | -Winvalid-noreturn |  |
| CodeSonar | 7.4p0 | **LANG.STRUCT.RFNR** | Return from noreturn |

#### Coding Standard 10

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

| **Noncompliant Code** |
| --- |
| The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. |
| #include <exception>  #include <fstream>  #include <string>    **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| #include <exception>  #include <fstream>  #include <string>    **void** f(**const** std::string &fileName) {    std::fstream file(fileName);  **if** (!file.is\_open()) {      // Handle error  **return**;    }    // ...    file.close();  **if** (file.fail()) {      // Handle error    }    std::terminate();  } |

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

| **Principles(s):**  2: Heed Compiler Warnings  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube C/C++ Plugin | 3.11 | **S2095** |  |
| Polyspace Bug Finder | R2023a | CERT C: Rule FIO42-C | Checks for resource leak (rule partially covered) |
| PC-lint Plus | 1.4 | **429** | Partially supported |
| Parasoft C/C++test | 2023.1 | **CERT\_C-FIO42-a** | Ensure resources are freed |

### 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.

Automation will enforce standards in several areas within the existing DevOps. What is automation script in DevOps? DevOps automation scripts are the tools used to automate the DevOps processes, allowing developers to focus on their core tasks and speed up software delivery. These automation scripts ensure fast testing and deployment of the software in a consistent, reliable, and repeatable way. Looking into the verify and testing phase, you can implement many automation tools. Monitor and detecting can have automatic alerts, which can help notify when problems occur.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | Low | Unlikely | Medium | P2 | 3 |
| STD-003-CPP | High | Likely | Medium | P18 | 1 |
| STD-004-CPP | High | Probable | Medium | P12 | 1 |
| STD-005-CPP | High | Likely | Medium | P18 | 1 |
| STD-006-CPP | Low | Unlikely | High | P1 | 3 |
| STD-007-CPP | Low | Probable | Medium | P4 | 3 |
| STD-008-CPP | High | Likely | Medium | P18 | 1 |
| STD-009-CPP | Medium | Unlikely | Low | P2 | 3 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | 3 |

### 

### Create Policies for Encryption and Triple A

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

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

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

| **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is encryption that is used to help protect data that is stored on a disk or backup media. |
| Encryption at flight | Encryption in flight is the encryption of data that moves over a network. This is especially important for those using open internet for transporting data, which is part of most public cloud implementations. |
| Encryption in use | In-Use Encryption not only encrypts the underlying data but analyzes data requests in real time and blocks suspicious requests. |

| **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of identifying a user and granting them access to the network. The server evaluates the credential data submitted by the user compared to the ones stored in the network's database. |
| Authorization | After authentication, the authorization process enforces the network policies, granular access control, and user privileges. It also establishes the tasks and activities that users can perform within those authorized resources. |
| Accounting | Accounting, the final process in the framework, is all about measuring what's happening within the network. Accounting helps in both security and operational evaluations. |

**\***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 | 09/10/23 | Module 3 Milestone | Mikaylah Blunt |  |
| 1.2 | 10/06/23 | Module 6 Milestone | Mikaylah Blunt |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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