**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 | Ensures that the input that is coming in can be trusted. The best way to do this is to assume that all inputs are a security threat. |
| 1. Heed Compiler Warnings | When the code is compiled, we must test it at the highest level of security and detect issues immediately. The code can show us which parts need to be changed to make it secure. |
| 1. Architect and Design for Security Policies | Following security policies can help you the architecture and design to be more secure. Although this can be a lot to ask for, you can make the design process easier by utilizing some designs that are already in place. |
| 1. Keep It Simple | Keeping the design simple will help the code to remain neat and easily maintainable. If the system were to be a huge, complex system, then it would be quite difficult to maintain. |
| 1. Default Deny | Using a default deny can be extremely helpful to make a software more secure. It allows an admin to make the final call on who can access certain data and who cannot. |
| 1. Adhere to the Principle of Least Privilege | This idea will ensure that the job is finished with the least amount necessary. As for the elevated tasks, the task should only be accessed at a small amount of time. Doing this allows the attacker to have less time to attack. |
| 1. Sanitize Data Sent to Other Systems | The code that isn’t being used should be removed so that hackers cannot use the code to get into the system. Without this, hackers could use injections to get into the system. |
| 1. Practice Defense in Depth | Defense in Depth will allow you to have a design with multiple layers of security. The idea is to cause an attacker to go through multiple layers to get into the system. |
| 1. Use Effective Quality Assurance Techniques | Effective QA allows you to see if the process of the system is operating correctly. |
| 1. Adopt a Secure Coding Standard | Using a standard will help you secure the software. Once a standard is in place, it can be easily changed with the more experience you have. |

### 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 | Prevents Out-Of-Range value. |

| **Noncompliant Code** |
| --- |
| Checks if a given value is within an acceptable range. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| Checks the value to ensure it can be represented by the enumeration. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** Buffer Overflow can cause results to not be specified. Although, data integrity could be effected in this case, rather than issues with the code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| Helix QAC | 2021.2 | C++3013 |  |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeratio |
| PRQA QA-C++ | 4.4 | 3013 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Used for valid integer ranges. |

| **Noncompliant Code** |
| --- |
| The example shows the two iterators that delimit into the same container. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<**int**> &c) {    std::for\_each(c.end(), c.begin(), [](**int** i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| The values are passed in order. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<**int**> &c) {    std::for\_each(c.begin(), c.end(), [](**int** i) { std::cout << i; });  } |

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

| **Principles(s):** Arbitrary code could be ran if buffer overflow is not fixed. |
| --- |

**Threat Level**

| **Threat Level Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | overflow\_upon\_dereference |  |
| Helix QAC | 2021.2 | C++3802 |  |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-CTR53-a  CERT\_CPP-CTR53-b | -Do not use an iterator range that isn't really a range -Do not compare iterators from different containers |
| PRQA QA-C++ | 4.4 | 3802 |  |
| PVS-Studio | 7.13 | V539, V662, V789 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Preventing to create a std::string from a null pointer. |

| **Noncompliant Code** |
| --- |
| A std::string object is created from the result of a call to std::getenv(). |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::**getenv**("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| The results of the call to std::getenv() are checked for null before the std::string object is contructed. |
| #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):** Random program termination is taking place. Leading to low severity. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | assert\_failure |  |
| Helix QAC | 2021.2 | C++4770, C++4771, C++4772, C++4773, C++4774 |  |
| Klocwork | 2021.1 | 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 |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Prevents the storing of an already-owned pointer value. |

| **Noncompliant Code** |
| --- |
| Two smart pointers are formed. When the local variable is destroyed, it deletes the value. This can result in an injection. |
| #include <memory>    void f() {  **int** \*i = new **int**;    std::shared\_ptr<**int**> p1(i);    std::shared\_ptr<**int**> p2(i);  } |

| **Compliant Code** |
| --- |
| The std::shared\_ptr objects are related to each other through copy construction. When p2 is destroyed, the use coun for the shared pointer value is decremented but still nonzero. Then, when p1 is destroyed, the use count for the shared pointer value is decremented to zero and the managed pointer is destroyed. |
| #include <memory>    void f() {    std::shared\_ptr<**int**> p1 = std::make\_shared<**int**>();    std::shared\_ptr<**int**> p2(p1);  } |

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

| **Principles(s):** Pointer is passed to function that is not matching. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | 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 |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| Parasoft C/C++test | 2021.1 | CERT\_C-STR31-a  CERT\_C-STR31-b  CERT\_C-STR31-c  CERT\_C-STR31-d  CERT\_C-STR31-e | Avoid accessing arrays out of bounds  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Avoid using unsafe string functions which may cause buffer overflows |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Exhaustively verified |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Prevents access to freed memory. |

| **Noncompliant Code** |
| --- |
| Vulnerability can be exploited after the dereference is deallocated. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| The allocated memory is not deallocated. |
| #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):** Allocated memory that is being read after it has been deallocated can lead to a terminated program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| Polyspace Bug Finder | R2021a | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| Helix QAC | 2021.2 | C++3225, C++3226, C++3227, C++3228, C++3229, C++4632 | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Static assertion to test the value of the expression. |

| **Noncompliant Code** |
| --- |
| Assert() macro is used to assert a property using a memory-mapped structure. |
| #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** |
| --- |
| Before processor statement is used to allow an insertion of constant expressions. |
| 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):** Software defects can be detected using static assertion. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axicion Bauhaus Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Compass/ROSE |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation. |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle used for all expressions. |

| **Noncompliant Code** |
| --- |
| f() or main() catch exceptions thrown by throwing\_func(). |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| All exceptions handled by the main entry point. |
| 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):** Terminating a program incorrectly can lead to issues that can have a major security impact. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | main-function-catch-all-early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 |  |
| Helix QAC | 2021.2 | C++4035, C++4036, C++4037 |  |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-ERR51-a,  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point. |
| Polyspace Bug Finder | R2021a | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| PRQA QA-C++ | 4.4 | 4035, 4036, 4037 |  |
| RuleChecker | 20.10 | Main-function-catch-all-early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-008-CPP | One Definition Rule. |

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

| **Compliant Code** |
| --- |
| Programmers intent will define the mitigation. |
| // 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):** Improper use of the QDR can lead to unsolved behaviors that can lead to service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | **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 |  |
| CodeSonar | 6.1p0 | **LANG.STRUCT.DEF.FDH LANG.STRUCT.DEF.ODH** | Function defined in header file Object defined in header file |
| Helix QAC | 2021.2 | C++1067, C++1509, C++1510 |  |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Partially implemented |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-DCL60-a, | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| Polyspace Bug Finder | R2021a | CERT C++: DCL60-CPP | Checks for inline constraints not respected (rule partially covered) |
| RuleChecker | 20.10 | **type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading** | Partially checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-009-CPP | Basic string uses references, iterators, and pointers. |

| **Noncompliant Code** |
| --- |
| Iterator loc. |
| #include <string>    void f(const std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();    for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

| **Compliant Code** |
| --- |
| Iterator loc is updated as a result to call to insert(). |
| #include <string>    void f(const std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();    for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      loc = email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

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

| **Principles(s):** Arbitrary code can be ran if there is a use of an invalid pointer, iterator, or reference. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++4746, C++4747,  C++4748, C++4749 |  |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-STR52-a, | Use valid references, pointers, and iterators to reference elements of a basic\_string |

**Coding Standard 10**

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-010-CPP | Input and Output does not alternate. |

| **Noncompliant Code** |
| --- |
| Code appends the data to the end of the file. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";    std::string str;    file >> str;  } |

| **Compliant Code** |
| --- |
| The std::basic\_istream<T>::seekg() function is called between the output and input. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    file << "Output some data";    std::string str;    file.seekg(0, std::ios::beg);    file >> str;  } |

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

| **Principles(s):** Undefined behavior is detected when input and output is not called properly. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-FI050 |  |
| Helix QAC | 2021.2 | C++4711, C++4712,  C++4713 |  |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-FI050-a, | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Finder | R2021a | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

### 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 can be created upon Build, by automating manual processes into a CI/CD pipeline. Automation makes the manual process significantly less.

Implement automated security tests and regression tests in QA. Perform an analysis of the data to ensure proper data.

Utilize tools to check code dependency vulnerabilities.

### 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 | High | P9 | L2 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Probable | Medium | P4 | L3 |
| STD-009-CPP | High | Probable | High | P6 | L2 |
| STD-010-CPP | High | Probable | High | P6 | L2 |
| STD-001-CPP | High | Likely | High | P9 | L2 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |

### 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 | Local server is used to encrypt the data. The data should be backed up as well. |
| Encryption at flight | Using a public key, the data should be fully up-to-date and secured through a library. |
| Encryption in use | Users roles will be confirmed and identified. Specific access will be granted to certain users. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Identifying a user, allowing them to access the data that they are entitled to. Password must. Be valid as well. |
| Authorization | After authentication, the user has specific access to certain data that they are authorized to see and use. |
| Accounting | Monitoring user events that have been input into the data. |

**\***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 |  |
| 1.5 | 04/09/2023 | Full Update on Document | Justin Parker | Justin Parker |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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