**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 | Validating all input is very beneficial in reducing vulnerabilities. It is important to validate data coming from untrusted sources, and even trusted sources, to ensure that malicious data isn’t being introduced to the system. |
| 1. Heed Compiler Warnings | Paying attention to the compiler warnings helps to catch vulnerabilities we may not have otherwise. Many times, they are not of the utmost importance, but sometimes they could lead to fixing a vulnerability. |
| 1. Architect and Design for Security Policies | Designing code to incorporate and follow security policies from the beginning, while still accomplishing what the program is intended to is very beneficial. This will save much work down the road and is way more efficient because vulnerabilities won’t have to be fixed after the fact. |
| 1. Keep It Simple | Having clean and concise code makes implementing security of the code much easier. Code that is more complex than needed is more likely to leave vulnerabilities open to attack. |
| 1. Default Deny | Default deny is a good policy because it denies all incoming traffic that is not expressly permitted. This reduces unnecessary traffic and the amount of points an attacker could target. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege is important as it limits access to only those who require it. By limiting access to those who need it, we decrease the chances that someone with malicious intentions gets entry into the system. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data that is sent to other systems helps ensure that no vulnerabilities or sensitive information is sent. Code sent with this information could lead to the system being exploited. |
| 1. Practice Defense in Depth | Defense in Depth is important is very important because it means having multiple layers of security. This is beneficial because one layer of security fails all is not lost because there are other layers to pick up the slack. Systems should always have more than one layer of security for this reason. |
| 1. Use Effective Quality Assurance Techniques | Effective Quality Assurance helps to find bugs and vulnerabilities before a system or code is put into production. Having a solid process of Quality Assurance is crucial, and incorporating aspects such as pen testing and independent security analysis can help to find vulnerabilities that might get missed by an internal team. |
| 1. Adopt a Secure Coding Standard | Having a standard for security in the code we produce is important in the sense it incorporates accountability as the code is being created. If we work to create secure code from the beginning, it is much easier to protect and keep secure in the long run. |

### 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] | Don’t cast an out-of-range enumeration value as it can cause unexpected behavior. |

| **Noncompliant Code** |
| --- |
| This code checks whether a value is inside of a given range of acceptable values. However, it casts to the enumeration type before being checked which could allow an unspecified value to be passed, resulting in unspecified/unexpected behavior. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  //error  }  } |

| **Compliant Code** |
| --- |
| The version of this code is compliant and checks the value before performing conversion to be sure that the conversion doesn’t result in unspecified value. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  if (enumVar < First || enumVar > Third) {  //error  }    EnumType enumVar = static\_cast<EnumType>(intVar);    } |

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

| **Principles(s):**  1. Validate input data – This applies because casting to an unspecified value doesn’t allow the input to be validated.  3. Architect and design code for security policies – Incorporate security policies form the beginning or design code that allows it to be implemented easily.  4. Keep it simple – this principle will likely apply to all coding standards. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion alters value  Cast alters value |
| Astree | 22.10 | cast.integer-to-enum | Partially checked |
| Parasoft C/C++test | 2022.2 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| RuleChecker | 22.10 | cast.integer-to-enum | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use valid references, pointers, and iterators to reference elements of a container |

| **Noncompliant Code** |
| --- |
| In the noncompliant example, pos is invalidated after the first call to insert() which results in unexpected behavior. |
| #include <deque>    **void** f(**const** **double** \*items, std::**size\_t** count) {  std::deque<**double**> d;  auto pos = d.begin();  **for** (std::**size\_t** i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| In the compliant version of this code, pos is assigned a valid iterator on each insertion which prevents the unexpected behavior from happening. |
| #include <deque>    **void** f(**const** **double** \*items, std::**size\_t** count) {  std::deque<**double**> d;  auto pos = d.begin();  **for** (std::**size\_t** i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 41.0);  }  } |

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

| **Principles(s):**  2. Heed compiler warnings  5. Default deny |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | ALLOC.UAF | Use Alter Free |
| Astree | 22.10 | overflow\_upon\_dereference |  |
| Parasoft C/C++ Test | 2022.2 | CERT\_CPP-CTR51-a | Do not modify a container while iterating over it. |
| Polyspace Bug Finder | R2023a | CERT C++: CTR51-CPP | Checks for use of invalid iterator |

#### 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** |
| --- |
| This example, which is noncompliant, shows a std::string object created from results of a call to std::gtenv(), but std::getenv() returns a null pointer on failure. Thus, undefined behavior can occur if the environment variable doesn’t exist or another error occurs. |
| #include <cstdlib>  #include <string>    **void** f() {  std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| In the compliant version of this code, the results of the call to std::getenv() are checked for null before the string object is created. |
| #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):**  3. Architect and design for security policies  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 | Assert\_failure | [Insert text.] |
| CodeSonar | 7.3p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |
| Polyspace bug finder | R2023a | CERT C++:STR51-CPP | Checks for string operations on null pointer |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Don’t write syntactically ambiguous declarations. |

| **Noncompliant Code** |
| --- |
| In this example an anonymous local variable of the type std::unique\_lock, is expected to lock and unlock the mutex by virtue of RAII. The syntax used in this example is ambiguous and thus thee mutex is never locked. |
| #include <mutex>    **static** std::mutex m;  **static** **int** shared\_resource;    **void** increment\_by\_42() {  std::unique\_lock<std::mutex>(m);  shared\_resource += 42;  } |

| **Compliant Code** |
| --- |
| In the compliant version, the lock is given an identifer and the converting constructor is called. |
| #include <mutex>    **static** std::mutex m;  **static** **int** shared\_resource;    **void** increment\_by\_42() {  std::unique\_lock<std::mutex> lock(m);  shared\_resource += 42;  } |

**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 | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE |  |
| Parasoft Jtest | 2022.2 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SonarQube | 6.7 | S2077  S3649 | SQL queries should not be vulnerable to injection attack |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly deallocate dynamically allocated resources. |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the resulting pointer is passed to operator::delete() resultingin undefined behavior because operator::delete() is attempting to free memory that was not returned by operator::new(). |
| #include <iostream>    **struct** S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {  alignas(**struct** S) **char** space[**sizeof**(**struct** S)];  S \*s1 = **new** (&space) S;    // ...    **delete** s1;  } |

| **Compliant Code** |
| --- |
| In this compliant version, the call to operator::delete() is removed and s1’s destructor is explicitly called. |
| #include <iostream>    **struct** S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    **void** f() {  alignas(**struct** S) **char** space[**sizeof**(**struct** S)];  S \*s1 = **new** (&space) S;    // ...    s1->~S();  } |

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

| **Principles(s):**  8. Practice Defense in Depth  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 | 22.10 | Invalid\_dynamic\_memory\_allocation  dangling\_pointer\_use |  |
| CodeSonar | 7.3p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM  ALLOC.LEAK | Free non-heap variable  Double free  Type mismatch  Leak |
| Parasoft Insure++ | [Insert text.] | [Insert text.] | Runtime detection |
| Polyspace Bug Finder | R2023a | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Checks for:  Invalid pointer deletion  Invalid free of pointer  Deallocation of previously deallocated pointer |

#### 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** |
| --- |
| This noncompliant example uses 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** |
| --- |
| In the compliant version a preprocessor conditional statement is used instead. |
| **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):**  9. Use effective quality assurance techniques. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.3p0 | customization | Users can implement a check that reports uses of assert() macro |
| 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 the exceptions thrown before main() begins executing |

| **Noncompliant Code** |
| --- |
| In the noncompliant example, neither f() or main() catch exceptions thrown by throwing\_func() and because no matching handler is found, std::terminate() is called |
| **void** throwing\_func() noexcept(**false**);    **void** f() {  throwing\_func();  }    **int** main() {  f();  } |

| **Compliant Code** |
| --- |
| In the compliant version, the main entry point handles all of the exceptions which ensures that the stack is unwound up to the main() function and allows 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 | Likely | Low | P9 | L2` |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Potentially-throwing-static-initialization | Partially checked |
| Clang | 3.9 | CertC++-ERR58 | Checked by clang-tidy |
| CodeSonar | 7.3p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-ERR58-a | Exception shall be raised only after start-up and before termination of the program |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Exclude user input from format strings |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the incorrect\_password() function accepts the name of the user as a string referenced by the user. This is an example of untrusted data that comes from an untrusted source. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    **void** incorrect\_password(**const** **char** \*user) {  **int** ret;  /\* User names are restricted to 256 or fewer characters \*/  **static** **const** **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + **sizeof**(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);  **if** (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  **if** (ret < 0) {  /\* Handle error \*/  } **else** **if** (ret >= len) {  /\* Handle truncated output \*/  }  **fprintf**(stderr, msg);  **free**(msg);  } |

| **Compliant Code** |
| --- |
| The compliant version replaces fprintf() with a call to fputs() which outputs msg directly to stderr without evaluating its content. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    **void** incorrect\_password(**const** **char** \*user) {  **int** ret;  /\* User names are restricted to 256 or fewer characters \*/  **static** **const** **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + **sizeof**(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);  **if** (msg == NULL) {  /\* Handle error \*/  }  ret = snprintf(msg, len, msg\_format, user);  **if** (ret < 0) {  /\* Handle error \*/  } **else** **if** (ret >= len) {  /\* Handle truncated output \*/  }  **fputs**(msg, stderr);  **free**(msg);  } |

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

| **Principles(s):**  4. Keep it simple  7. Sanitize data sent from other systems |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| Coverity | 2017.07 | TAINTED\_STRING | Implemented |
| Parasoft C/C++test | 2022.2 | CERT\_C-FIO30-a  CERT\_C-FIO30-b  CERT\_C-FIO30-c | Avoid calling functions printf/wprintf with only one argument other than string constant.  Avoid using functions fprintf/fwprintf with only two parameters when second parameter is a variable.  Never use unfiltered data from an untrusted user as the format parameter. |
| Polyspace Bug Finder | R2023a | CERT C: Rule FIO30-C | Checks for tainted string format (rule partially uncovered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Do not reuse variable names in subscopes |

| **Noncompliant Code** |
| --- |
| This noncompliant example declares msg identifier at file scope and reuses the same identifier to declare a character array local to the report\_error() function. |
| #include <stdio.h>    **static** **char** msg[100];  **static** **const** **size\_t** msgsize = **sizeof**( msg);    **void** report\_error(**const** **char** \*str) {  **char** msg[80];  snprintf(msg, msgsize, "Error: %s\n", str);  /\* ... \*/  }    **int** main(**void**) {  /\* ... \*/  report\_error("some error");    **return** 0;  } |

| **Compliant Code** |
| --- |
| This example is compliant and uses different and more descriptive variable names. |
| #include <stdio.h>    **static** **char** message[100];  **static** **const** **size\_t** message\_size = **sizeof**( message);    **void** report\_error(**const** **char** \*str) {  **char** msg[80];  snprintf(msg, **sizeof**( msg), "Error: %s\n", str);  /\* ... \*/  }    **int** main(**void**) {  /\* ... \*/  report\_error("some error");    **return** 0;  } |

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

| **Principles(s):**  **2. Heed compiler warnings**  **4. Keep it simple** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| B::Lint | 5.0 | .\* masks earlier declaration in same scope | Implemented |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Value returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| In the non-compliant example, the developer forgot to return the input value for positive input, thus not all the code paths return a value. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;  }  } |

| **Compliant Code** |
| --- |
| The compliant example shows that all code paths return a value. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;  }  **return** a;  } |

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

| **Principles(s):**  4. Keep it simple  9. Use effective quality assurance techniques. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Return-implicit | Fully checked |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as function-try-blocks |
| CodeSonar | 7.3p0 | LANG.STRUCT.MRS | Missing return statement |
| Polyspace Bug Finder | R2023a | CERT C++:MSC52-CPP | Checks for missing return statements (rule partially 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.

[Insert your written explanations here.]

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

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption for data at rest is the process of encoding data as it is written into storage and decrypting it as it is pulled from storage. Using a symmetric encryption key when the data is written into storage protects it from unauthorized access by anyone who does not have that key. It should be used anytime data is sensitive and would cause harm if accessed by unauthorized people. |
| Encryption at flight | Encryption of data in flight is the process of encoding data as it is being transmitted in some fashion. How you will be transferring data will determine how to apply encryption. When using a web browser always utilize secure protocols, when sending emails always encrypt before sending and use digital signatures |
| Encryption in use | Encryption in use is the process of protecting data as it is utilized in memory. The main way of doing this is by utilizing password protected profiles, as they protect the memory of each user for the data stored in memory for that profile could be used to compromise their data in rest or flight. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Process is used to prove who a user is by verifying their user id, password, and any other means of verification of that user’s identity. Users must be authenticated at log in before gaining access to the system. |
| Authorization | Once a user is authenticated, they may be given authorization to various aspects of the system depending on their role and granted permissions. For example, users must have the authorization to add new users to the system. |
| Accounting | Accounting after authentication and authorization is important to keep track of who does what within the system. This way the activities of all users within the system are documented, such as changes to the database and files accessed by the user. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 03/25/2023 | Added coding standards and principles | Robert Sanford | [Insert text.] |
| 2.1 | 04/01/2023 | Final Revision of coding standards and principles | Robert Sanford | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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