**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 input data is an essential phase in software development for ensuring the system's integrity and security. Verifying the accuracy, completeness, and conformance of data entered by users or received from external sources. By incorporating input validation, developers can avoid problems such as data corruption, injection attacks, and system crashes caused by incorrect or malicious data. Validation of input data contributes to system reliability and helps preserve data integrity. |
| 1. Heed Compiler Warnings | Compiler warnings are messages generated by the compiler that alert developers to potential issues or questionable coding practices in their code. Paying attention to these warnings allows developers to identify and address possible bugs, uninitialized variables, unused variables, or other problematic code constructs before they manifest as runtime errors or unexpected behavior. By addressing compiler warnings promptly, developers can improve code quality, enhance program stability, and reduce the likelihood of encountering bugs during runtime. |
| 1. Architect and Design for Security Policies | Security policy architecture and design entail embedding security measures into the overall design and structure of a system. Identifying and comprehending the system's specific security requirements and objectives, defining access restrictions and permissions, implementing secure communication protocols, and assuring data confidentiality, integrity, and availability are all part of this process. By including security policies in the design process early on, developers can construct a system that is intrinsically secure, reducing potential vulnerabilities and safeguarding sensitive data from unauthorized access or misuse. |
| 1. Keep It Simple | A fundamental rule of software development is to keep the code simple. Writing code that delivers its intended functionality without adding needless complexity entails writing it in a way that is simple, straightforward, and comprehensible. Simple code is easier to refactor, debug, read, and maintain. It encourages improved developer cooperation, lowers the risk of introducing defects, and improves the overall efficiency and quality of the code. |
| 1. Default Deny | The idea of limiting access by default until it is expressly allowed is known as default denial. In other words, only authorized and verified activities are permitted, and all other requests or actions are first denied. The default denies strategy helps developers implement strong access control and reduce potential security flaws by ensuring that malicious or unauthorized activity is avoided until specifically allowed. The dangers brought on by unauthorized access, injection attacks, and other security threats are lessened because of this approach. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege is a security principle that states a user or process should only be granted the privileges necessary for it to execute its intended tasks. This involves only granting users or processes the permissions and access rights they require. By adhering to the principle of least privilege, developers can limit the potential damage caused by compromised accounts or malicious actions, reduce the attack surface, and improve overall system security by assigning privileges based on a user's need to know. |
| 1. Sanitize Data Sent to Other Systems | Before data is transmitted to external systems or APIs, it must be cleaned and validated. It involves eliminating or neutralizing potentially harmful or unexpected characters, authenticating input formats, and ensuring data integrity. By sanitizing data, developers can prevent injection attacks, data corruption, and unauthorized access to sensitive information, thereby improving the security and dependability of data transmission between systems. |
| 1. Practice Defense in Depth | Defense in depth is a security strategy that involves implementing multiple layers of defense mechanisms to protect against security attacks. Instead of relying on a single security measure, it creates a series of barriers that an attacker must overcome to gain unauthorized access or cause damage. Each layer provides a unique level of protection, and if one layer fails, the remaining layers serve as backups, reducing the overall risk and impact of an attack |
| 1. Use Effective Quality Assurance Techniques | Adopting a secure coding standard entails adhering to a set of guidelines and best practices during software development to reduce security risks and vulnerabilities. It includes input validation, error management, secure authentication and authorization mechanisms, secure data storage and transmission, and so on. By adhering to a secure coding standard, developers can reduce the likelihood of introducing common security flaws like SQL injection, cross-site scripting, and buffer overflows. It ensures that the software is developed with security in mind, making it more resistant to attacks and safeguarding sensitive information. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard entails adhering to a set of guidelines and best practices during software development to reduce security risks and vulnerabilities. It includes input validation, error management, secure authentication and authorization mechanisms, secure data storage and transmission, and so on. By adhering to a secure coding standard, developers can reduce the likelihood of introducing common security flaws like SQL injection, cross-site scripting, and buffer overflows. It ensures that the software is developed with security in mind, making it more resistant to attacks and safeguarding sensitive information. |

### 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** | **Integers (INT)** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to check whether a given value is within the range of acceptable enumeration values. However, it is doing so after casting to the enumeration type, which may not be able to represent the given integer value. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| This compliant solution checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

| **Principles(s):**  Validating Input Data – ensuring proper inputs  Architect and Design for Security Policies – building code to prevent vulnerabilities  Keep it simple –keeping code as lightweight as possible is best practice |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **cast-integer-to-enum** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-INT50** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.CAST.COERCE**  **LANG.CAST.VALUE** | Coercion Alters Value  Cast Alters Value |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3013** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-INT50-a** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.25 | V1016 |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | cast-integer-to-enum | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Containers (CTR)** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Guarantee that container indices and iterators are within the valid range |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows a function, insert in table(), that has two int parameters, pos and value, both of which can be influenced by data originating from untrusted sources. The function performs a range check to ensure that pos does not exceed the upper bound of the array, specified by tableSize, but fails to check the lower bound. Because pos is declared as a (signed) int, this parameter can assume a negative value, resulting in a write outside the bounds of the memory referenced by table. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, int pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the parameter pos is declared as size\_t, which prevents the passing of negative arguments. |
| #include <cstddef>    void insert\_in\_table(int \*table, std::size\_t tableSize, std::size\_t pos, int value) {  if (pos >= tableSize) {  // Handle error  return;  }  table[pos] = value;  } |

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

| **Principles(s):**  Heed compiler warnings – always pay attention to warnings as they are there for a reason  Architect and Design for Security Policies – building code to prevent vulnerabilities  Keep it simple – always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques – making tests that are as effective as possible  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **overflow\_upon\_dereference** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.MEM.BO LANG.MEM.BU LANG.MEM.TO LANG.MEM.TU LANG.MEM.TBA LANG.STRUCT.PBB LANG.STRUCT.PPE LANG.STRUCT.PARITH** | Buffer overrun  Buffer underrun  Type overrun  Type underrun  Tainted buffer access  Pointer before beginning of object  Pointer past end of object  Pointer Arithmetic |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3139, C++3140**  **DF2891** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **ABV.ANY\_SIZE\_ARRAY** **ABV.GENERAL** **ABV.GENERAL.MULTIDIMENSION** **ABV.STACK** **ABV.TAINTED** **SV.TAINTED.ALLOC\_SIZE** **SV.TAINTED.CALL.INDEX\_ACCESS** **SV.TAINTED.CALL.LOOP\_BOUND** **SV.TAINTED.INDEX\_ACCESS** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-CTR50-a** | Guarantee that container indices are within the valid range |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: CTR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcctr50cpp.html) | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.25 | [**V781**](https://pvs-studio.com/en/docs/warnings/v781/) |  |
|  |  |  |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Characters and Strings (STR)** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| Because the input is unbounded, the following code could lead to a buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The best solution for ensuring that data is not truncated and for guarding against buffer overflows is to use std::string  instead of a bounded array, as in this compliant solution. |
| #include <iostream>  #include <string>    void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):**  Heed compiler warnings – always pay attention to warnings as they are there for a reason  Architect and Design for Security Policies – building code to prevent vulnerabilities  Keep it simple – always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques – making tests that are as effective as possible  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **stream-input-char-array** | [Insert text.] |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | Partially checked + soundly supported |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++5216**  **DF2835, DF2836, DF2839,** | No space for null terminator  Buffer overrun  Type overrun |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **NNTS.MIGHT** **NNTS.TAINTED** **NNTS.MUST** **SV.UNBOUND\_STRING\_INPUT.CIN** | [Insert text.] |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **stream-input-char-array** |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3519**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3519) | Partially checked |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Characters and Strings (STR)** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Range check element access |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the value returned by the call to get\_index() may be greater than the number of elements stored in the string, resulting in undefined behavior. |
| #include <string>    extern std::size\_t get\_index();    void f() {  std::string s("01234567");  s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| This compliant solution uses the std::basic\_string::at() function, which behaves in a similar fashion to the index operator[] but throws a std::out\_of\_range exception if pos >= size(). |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();    void f() {  std::string s("01234567");  try {  s.at(get\_index()) = '1';  } catch (std::out\_of\_range &) {  // Handle error  }  } |

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

| **Principles(s):**  Validating Input Data – ensuring proper inputs  Architect and Design for Security Policies – building code to prevent vulnerabilities  Keep it simple – always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques – making tests that are as effective as possible  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | 22.10 | **assert\_failure** |  |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 7.3p0 | **LANG.MEM.BO** **LANG.MEM.BU** **LANG.MEM.TBA** **LANG.MEM.TO** **LANG.MEM.TU** | Buffer overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 2023.1 | **C++3162, C++3163, C++3164, C++3165** | [Insert text.] |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.2 | **CERT\_CPP-STR53-a** | Guarantee that container indices are within the valid range |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | R2023a | [CERT C++: STR53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr53cpp.html) |  |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | 22.10 | **assert\_failure** | Checks for:   * Array access out of bounds * Array access with tainted index * Pointer dereference with tainted offset   Rule partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Memory Management (MEM)** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not store an already-owned pointer value in an unrelated smart pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, two unrelated smart pointers are constructed from the same underlying pointer value. When the local, automatic variable p2 is destroyed, it deletes the pointer value it manages. Then, when the local, automatic variable p1 is destroyed, it deletes the same pointer value, resulting in a double-free vulnerability. |
| #include <memory>    void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::shared\_ptr objects are related to one another through copy construction. When the local, automatic variable p2 is destroyed, the use count for the shared pointer value is decremented but still nonzero. Then, when the local, automatic variable p1 is destroyed, the use count for the shared pointer value is decremented to zero, and the managed pointer is destroyed. This compliant solution also calls std::make\_shared() instead of allocating a raw pointer and storing its value in a local variable. |
| #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):**  Validating Input Data – ensuring proper inputs  Architect and Design for Security Policies – building code to prevent vulnerabilities  Keep it simple – always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques – making tests that are as effective as possible  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **dangling\_pointer\_use** |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MEM56** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF4721, DF4722, DF4723** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MEM56-a** | Do not store an already-owned pointer value in an unrelated smart pointer |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: MEM56-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem56cpp.html) | Checks for use of already-owned pointers (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.25 | [**V1006**](https://pvs-studio.com/en/docs/warnings/v1006/) |  |

#### 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** |
| --- |
| noncompliant code uses the assert() macro to assert a property concerning a memory-mapped  structure that is essential for the code to behave correctly. The use of the runtime assertion is  better than nothing, it needs to be placed in a function and executed. |
| #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** |
| --- |
| Static assertions allow incorrect assumptions to be diagnosed at compile time instead of resulting in a silent  malfunction or runtime error |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  static\_assert(sizeof(struct timer) == sizeof(unsigned char)  + sizeof(unsigned int) + sizeof(unsigned int), |

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

| **Principles(s):**  Validating Input Data – ensuring proper inputs  Architect and Design for Security Policies – building code to prevent vulnerabilities  Sanitize data sent to other systems – keeping data sent to other systems as only required/authorized  Keep it simple – always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques – making tests that are as effective as possible  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
|  |  |  |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-DCL03 | Checked by clang-tidy |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Users can implement a custom check that reports uses of the assert() macro |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | (customization) | 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 |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Fully implemented |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | CC2.DCL03 | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Checked by clang-tidy |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Exceptions and Error Handling (ERR)** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, 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** |
| --- |
| In this compliant solution, 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):** Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
|  |  |  |  |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **MISRA.CATCH.ALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  | **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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | 2022.2 | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | R2023a | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Miscellaneous (MSC)** |
| --- | --- | --- |
| Return Value | [STD-008-CPP] | Value-returning functions must return a value from all exit paths |

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

| **Compliant Code** |
| --- |
| In this compliant solution, all code paths now 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):**  Architect and Design for Security Policies – building code to prevent vulnerabilities  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-implicit** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-MSC52** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | **-Wreturn-type** | Does not catch all instances of this rule, such as function-try-blocks |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.MRS** | Missing return statement |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF2888** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **FUNCRET.GEN**  **FUNCRET.IMPLICIT** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **2 D, 36 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  | **CERT\_CPP-MSC52-a** | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | 2022.2 | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | R2023a | [**S935**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-935) |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 4.10 | [**V591**](https://pvs-studio.com/en/docs/warnings/v591/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 7.25 | **return-implicit** | Fully checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Input Output (FIO)** |
| --- | --- | --- |
| I/O | [STD-009-CPP] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. 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. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not 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;  }  // ...  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):**  Architect and Design for Security Policies – building code to prevent vulnerabilities  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
|  |  |  |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **ALLOC.LEAK** | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **DF4786, DF4787, DF4788** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **RH.LEAK** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-FIO51-a** | Ensure resources are freed |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) |  |  | Runtime detection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: FIO51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcfio51cpp.html) | Checks for resource leak (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Exceptions and Error Handling (ERR)** |
| --- | --- | --- |
| String Errors | [STD-010-CPP] | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, multiple numeric values are converted from the standard input stream. However, if the text received from the standard input stream cannot be converted into a numeric value that can be represented by an int, the resulting value stored into the variables i and j may be unexpected. |
| #include <iostream>    void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| In this compliant solution, exceptions are enabled so that any conversion failure results in an exception being thrown. However, this approach cannot distinguish between which values are valid and which values are invalid and must assume that all values are invalid. Both the badbit and failbit flags are set to ensure that conversion errors as well as loss of integrity with the stream are treated as exceptions. |
| #include <iostream>    void f() {  int i, j;    std::cin.exceptions(std::istream::failbit | std::istream::badbit);  try {  std::cin >> i >> j;  // ...  } catch (std::istream::failure &E) {  // Handle error  }  } |

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

| **Principles(s):** Architect and Design for Security Policies – building code to prevent vulnerabilities  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
|  |  |  |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR62** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **BADFUNC.ATOF BADFUNC.ATOI BADFUNC.ATOL BADFUNC.ATOLL** | Use of atof  Use of atoi  Use of atol  Use of atoll |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3161** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **CERT.ERR.CONV.STR\_TO\_NUM** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-ERR62-a** | The library functions atof, atoi and atol from library stdlib.h shall not be used |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

I believe the pre-production phase's top priority should be to automate the policy's standards enforcement. Prior to the commencement of production, ensuring that everything is fully operational. If we are to begin production, this system must be free of any form of exposed security flaw. Regarding Production, we must ensure that Transition and Health Checks occur frequently. During the preproduction phase, security flaws are examined and their nature is determined. The Production phase is a method for preventing these security vulnerabilities from occurring, as well as preventing the occurrence of those that have already been discovered. Together, they ensure that the level of security is maintained at a high level.

### 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 | P4 | L3 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Unlikely | Medium | P6 | L2 |
| 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 | Medium | Probable | Medium | P8 | L2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### 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 | At-rest encryption, also referred to as data encryption at rest, involves encrypting data when it is stored or saved on a physical or digital storage medium, such as hard drives, solid-state drives, databases, or cloud storage services. It ensures that even if the storage media are compromised, the encrypted data remains unreadable without the appropriate decryption key or credentials.  At-rest encryption can be implemented through various methods, such as full disk encryption (FDE), where the entire storage device is encrypted, or file-level encryption (FLE), where individual files or specific data elements are encrypted. Encryption algorithms like AES (Advanced Encryption Standard) are commonly used to secure data at rest.  The policy of at-rest encryption should be applied to protect sensitive data whenever it is stored or saved, regardless of the storage medium. It is crucial for securing confidential information such as personally identifiable information (PII), financial records, trade secrets, intellectual property, and any other sensitive data that may be vulnerable to unauthorized access or theft. |
| Encryption at flight | In-flight encryption, also known as transit encryption, is the process of encrypting data while it is being transmitted between two systems over a network. It ensures that data remains secure and confidential while in transit. In-flight encryption is typically applied to various communication protocols, such as HTTPS (HTTP over SSL/TLS), VPN (Virtual Private Network), and SSH (Secure Shell). These protocols use encryption algorithms to protect the data during transmission, making it difficult for attackers to eavesdrop or tamper with the information.  The policy of in-flight encryption should be applied whenever data is being transmitted over a network, especially when it involves sensitive or confidential information. It is particularly important for protecting data during internet transactions, online banking, email communications, file transfers, and any other network-based data exchanges. |
| Encryption in use | In-use encryption, also known as runtime encryption or application-level encryption, focuses on protecting data while it is actively being processed or used by an application or a system. It involves encrypting data within the application's memory, ensuring that it remains secure even if the underlying infrastructure is compromised.  In-use encryption can be implemented through techniques like secure enclaves (e.g., Intel SGX) or application-level encryption libraries, which provide encryption and decryption capabilities within the application's code. This type of encryption helps protect data during computation or when it is temporarily stored in memory.  The policy of in-use encryption should be applied whenever sensitive data is being actively processed or used by an application or system. It is particularly important for applications handling sensitive data, such as financial systems, healthcare systems, and other scenarios where data confidentiality and integrity are critical. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying the identity of a user, system, or device attempting to access a resource or service. It ensures that only authorized entities are granted access. Authentication methods include passwords, biometrics (such as fingerprints or facial recognition), smart cards, or multifactor authentication (combining multiple authentication factors for increased security).  How it is used: Authentication mechanisms are employed at various points of entry, such as login screens, network gateways, or physical access control systems. They require users or devices to provide valid credentials or proof of identity before being granted access to protected resources.  Why it is necessary: Authentication is essential to prevent unauthorized access and protect sensitive data and systems from potential threats. It ensures that only legitimate users or devices with proper authorization can gain entry, reducing the risk of unauthorized data breaches or system compromise.  When the policy applies: The policy of authentication applies whenever there is a need to verify the identity of users, systems, or devices accessing resources. It should be implemented for applications, networks, databases, or any other system that stores or processes sensitive information. |
| Authorization | Authorization involves granting or restricting access rights and permissions to authenticated users or entities based on their assigned privileges. It defines what actions a user or system is allowed to perform within a given system or resource.  How it is used: Authorization mechanisms control access to various resources, such as files, folders, databases, or functionalities within an application. Access control lists (ACLs), role-based access control (RBAC), or attribute-based access control (ABAC) are commonly used to enforce authorization policies.  Why it is necessary: Authorization ensures that authenticated users are granted access only to the resources they are authorized to use, based on their roles, responsibilities, or privileges. It helps prevent unauthorized activities, data tampering, or misuse of resources by limiting access to specific users or user groups.  When the policy applies: The policy of authorization applies whenever there is a need to manage and control access to resources. It should be implemented in conjunction with authentication to ensure that authenticated entities are granted appropriate access rights based on their authorization level. |
| Accounting | Accounting, also referred to as auditing, focuses on tracking and recording user activities and system events for monitoring, analysis, and accountability purposes. It involves collecting and storing logs and audit trails that capture information about user actions, resource access, system configurations, and security-related events.  How it is used: Accounting mechanisms generate audit logs, which record critical information such as user logins, access attempts, file modifications, system changes, and security incidents. These logs can be analyzed for detecting anomalies, investigating security breaches, or ensuring compliance with regulations.  Why it is necessary: Accounting provides a means for accountability and transparency. It helps organizations identify security incidents, track user activities, detect unauthorized access attempts, and investigate potential breaches. It also assists in meeting regulatory requirements and establishing a baseline for security monitoring and incident response.  When the policy applies: The policy of accounting applies whenever there is a need to monitor, record, and analyze user activities, system events, or security-related incidents. It should be implemented across various systems, applications, and network devices to ensure comprehensive visibility and auditing capabilities. |

**\***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

**Data Type**

1- Validating Input Data – ensuring proper inputs

2- Heed compiler warnings – always pay attention to warnings as they are there for a reason

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

4- Keep it simple –keeping code as lightweight as possible is best practice

Data Value

2- Heed compiler warnings – always pay attention to warnings as they are there for a reason

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

4- Keep it simple – always applies as keeping code as lightweight as possible is best practice

9- Use Effective Quality Assurance Techniques – making tests that are as effective as possible

10- Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

String Correctness

2- Heed compiler warnings – always pay attention to warnings as they are there for a reason

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

4- Keep it simple – always applies as keeping code as lightweight as possible is best practice

9- Use Effective Quality Assurance Techniques – making tests that are as effective as possible

10-Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

SQL Injection

1- Validating Input Data – ensuring proper inputs

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

4- Keep it simple – always applies as keeping code as lightweight as possible is best practice

9- Use Effective Quality Assurance Techniques – making tests that are as effective as possible

10- Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

Memory Protection

1- Validating Input Data – ensuring proper inputs

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

4- Keep it simple – always applies as keeping code as lightweight as possible is best practice

9- Use Effective Quality Assurance Techniques – making tests that are as effective as possible

10- Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

Assertions

1- Validating Input Data – ensuring proper inputs

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

7- Sanitize data sent to other systems – keeping data sent to other systems as only required/authorized

4- Keep it simple – always applies as keeping code as lightweight as possible is best practice

9- Use Effective Quality Assurance Techniques – making tests that are as effective as possible

10 -Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

Exceptions

2- Heed compiler warnings – always pay attention to warnings as they are there for a reason

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

4- Keep it simple – always applies as keeping code as lightweight as possible is best practice

9- Use Effective Quality Assurance Techniques – making tests that are as effective as possible

10- Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

Return Value

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

10- Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

I/O

2- Heed compiler warnings – always pay attention to warnings as they are there for a reason

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

10- Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

String Errors

2- Heed compiler warnings – always pay attention to warnings as they are there for a reason

3- Architect and Design for Security Policies – building code to prevent vulnerabilities

10- Adopt a secure coding standard – making security a priority helps prevent vulnerabilities

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 | 5-20-2023 | Added coding Standards | James Carver | [Insert text.] |
| 3.0 | 6-10-2023 | Added Risk Assessments | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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