**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 | Perform input validation to ensure data is formatted correctly and within the expected ranges of a system. This helps to work against attacks such as SQL-injection. Additionally, input validation helps to prevent invalid data from entering a system by checking it as it enters. |
| 1. Heed Compiler Warnings | Pay attention to compiler warnings and correct them when possible. These warnings include information such as possible loss of data when converting data types and use of depreciated methods. These types of warnings can result in unexpected outcomes and present security risks. |
| 1. Architect and Design for Security Policies | Security policies should be present and accounted for when architecting and designing software. Software should be designed to uphold and carry out security principals rather than be an afterthought. |
| 1. Keep It Simple | Designs should be as simple as reasonably possible. This is because unnecessary complexity increases the possibility of errors and issues. Additionally, the increase in complexity also increases the difficulty of add security measures as they have to be just as complex. |
| 1. Default Deny | By default, access and privilege should be denied unless they meet the correct permissions. This helps to prevent accidental access and slow down attackers who find their way into a system. |
| 1. Adhere to the Principle of Least Privilege | All processes should be completed with the least number of privileges necessary. Any additional privileges should only be accessed for as long as they are needed. This helps to reduce the time and chances attackers have to utilize those privileges. |
| 1. Sanitize Data Sent to Other Systems | Sensitive data should be removed, masked, or encrypted before sending data to other systems. This helps to prevent situations where data is leaked because the system receiving the data is attacked and compromised. |
| 1. Practice Defense in Depth | Multiple defense measures should be used. This helps to minimize and prevent damages from a successful attack as if one measure fails another measure can help reduce the damages caused by the failure. |
| 1. Use Effective Quality Assurance Techniques | Strong quality assurance techniques can help identify and prevent possible exploits before they become an issue. Techniques such as code audits and penetration testing are proven quality assurance techniques to implement. |
| 1. Adopt a Secure Coding Standard | Secure coding standards are specifically designed to reduce security risks. Adopting and implementing one gives the team clear guidelines and practices to meet that will help improve security within a system. |

### 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** | **Use standard containers over raw arrays** |
| --- | --- | --- |
| **Data Type** | [DAT-001-CPP] | Standard containers are pre-tested, easy to understand, and should lead to predictable results reducing unexpected errors. |

| **Noncompliant Code** |
| --- |
| In this example, numbers is created using a raw array |
| int numbers[10]; |

| **Compliant Code** |
| --- |
| In this example, numbers is created using a standard vector. |
| std::vector<int> numbers; |

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

| **Principles(s):** 10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| N/A |  |  |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure division and remainder operations do not divide by zero** |
| --- | --- | --- |
| **Data Value** | [DAV-001-CPP] | Integer values my unexpectantly equal zero usually after performing arithmetic. We must ensure that divide by zero errors do not occur. |

| **Noncompliant Code** |
| --- |
| In this example, the limits of the signed ints are checked but not if divisor (s\_b) is not zero. |
| #include <limits.h>    void func(signed int s\_a, signed int s\_b) {  signed int result;  if ((s\_a == INT\_MIN) && (s\_b == -1)) {  /\* Handle error \*/  } else {  result = s\_a / s\_b;  }  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| In this example, the limits of the signed ints are checked along with confirming that the divisor (s\_b) is not zero. |
| #include <limits.h>    void func(signed int s\_a, signed int s\_b) {  signed int result;  if ((s\_b == 0) || ((s\_a == INT\_MIN) && (s\_b == -1))) {  /\* Handle error \*/  } else {  result = s\_a / s\_b;  }  /\* ... \*/  } |

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

| **Principles(s):** 10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | int-division-by-zero  int-modulo-by-zero | Fully Checked |
| CodeSonar | 8.1p0 | LANG.ARITH.DIVZERO  LANG.ARITH.FDIVZERO | Division by zero  Float Division By Zero |
| Polyspace Bug Finder | R2024a | CERT C: Rule INT33-C | Checks for:  Integer division by zero  Tainted division operand  Tainted modulo operand |
| Cppcheck | 2.15 | zerodiv  zerodivcond |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Ensure storage for strings is large enough** |
| --- | --- | --- |
| **String Correctness** | [STC-001-CPP] | Assigning a string length larger than the size of a character array will result in a buffer overflow. Therefore, we must ensure there is enough storage for strings. |

| **Noncompliant Code** |
| --- |
| In the below example the input can be larger than the 12 char limit set by buf resulting in buffer overflow. |
| void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| In this example, strings are used instead of a limited array to ensure that the inputs can be held. |
| #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):** 10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stream-input-char-array | Partially checked + soundly supported |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| [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 |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Use prepared statements** |
| --- | --- | --- |
| **SQL Injection** | [SQL-001-CPP] | In the event of attackers attempting to use SQL-injection, using prepared statements should force the query to search for the literal input given by the user and avoid a x=x attack. |

| **Noncompliant Code** |
| --- |
| In this example, Input is taken as is without looking for and removing any invalid or dangerous characters. |
| int main(){  std::string username;  std::cin >> username;  std::string sql = "SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME= ";  sql.append(username);  …  } |

| **Compliant Code** |
| --- |
| In this example, a prepared statement is used to catch the users input so that is entered as a literal string. |
| Int main(){  …  sql::PreparedStatement \*pstmt;  std::string username;  std::cin >> username;  …  // Prepare the statement  pstmt = con->prepareStatement("SELECT ID FROM USERS WHERE NAME(name) VALUES (?)");  pstmt ->setInt(1, 1);  pstmt ->setString(2, username);  pstmt ->execute();  } |

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

| **Principles(s):**  1: Validate Input Data – Here we are ensuring that the input is correctly formatted for the system.  **3:** Architect and Design for Security Policies – This is a security measure we are accounting for within the design of the system.  8: Practice Defense in Depth – This is one of multiple points of defense used to prevent attackers from accessing data.  7: Sanitize Data Sent to Other Systems – This is here to format data before sending to a SQL system.  10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [SQL injection](https://www.mathworks.com/help/bugfinder/ref/sqlinjection.html) | This checker is deactivated in a default Polyspace. For the issue to be detected, the checker must be enabled explicitly using the option -checkers SQL\_INJECTION. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | [MEM-001-CPP] | Attempting to access memory that has been deallocated results in unpredictable/undefined behavior that poses a risk to the system and its security. |

| **Noncompliant Code** |
| --- |
| In this example, memory is deleted then access is attempted resulting in undefined behavior. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| In this example, memory is not deleted until it is no longer needed. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):** 10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **dangling\_pointer\_use** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | **USE\_AFTER\_FREE** | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MEM50-a** | Do not use resources that have been freed |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Only use assertions for debugging** |
| --- | --- | --- |
| **Assertions** | [AST-001-CPP] | Since assertions abort a program when failed, this may result in resources being improperly handled when the program closes. As a result, assertions should only be used for debugging. |

| **Noncompliant Code** |
| --- |
| In this example, using assert results in the program aborting. |
| #include <assert.h>  int main() {  int t = 2    assert((t + 3) == 4);  …  } |

| **Compliant Code** |
| --- |
| In this example, an if statement has been used instead of the assert giving the program a chance to handle the situation rather then aborting the program. |
| int main() {  int t = 2    if((t + 3) == 4){  …  }  …  } |

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

| **Principles(s):** 10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| N/A |  |  |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all exceptions** |
| --- | --- | --- |
| **Exceptions** | [EXC-001-CPP] | If an exception is thrown without a handler to catch it, it will result in the uncontrolled termination of the program. This can improperly manage resources. |

| **Noncompliant Code** |
| --- |
| In this example, both the f and main functions fail to catch exceptions resulting in the termination of the program. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this example, the try catch in the main function handles the exception raised by the f function allowing the program to end in a controlled manner should it trigger. |
| 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):**  2: Heed Compiler Warnings – Should there be an instance of an exception not caught, it may be noticed by the compiler which should be corrected.  10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | Low | 2 |

**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) | 8.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not return from a function declared [[noreturn]]** |
| --- | --- | --- |
| Miscellaneous | [MIS-001-CPP] | When a function is declared no return and is able to return a value, the function will result in undefined behavior |

| **Noncompliant Code** |
| --- |
| If the value 0 is passed to the f function then the code will pass the if and else if checks and reach an implicit return. |
| #include <cstdlib>    [[noreturn]] void f(int i) {  if (i > 0)  throw "Received positive input";  else if (i < 0)  std::exit(0);  } |

| **Compliant Code** |
| --- |
| The code is unable to return or reach an implicit return. |
| #include <cstdlib>    [[noreturn]] void f(int i) {  if (i > 0)  throw "Received positive input";  std::exit(0);  } |

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

| 10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **invalid-noreturn** | Fully checked |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-MSC53-a** | Never return from functions that should not return |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.RFNR** | Return from noreturn |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: MSC53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc53cpp.html) | Checks for [[noreturn]] functions returning to caller (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Value returning functions must return a value from all exits** |
| --- | --- | --- |
| Miscellaneous | [MIS-002-CPP] | Functions returning values should return values from all paths, otherwise it may result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this example, integer inputs 0 or larger are unaccounted for, resulting in unexpected behavior. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| In this example, non-negative inputs are accounted for as anything that does not meet the initial if is returned. |
| 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):** 2: Heed Compiler Warnings – While not all instances may result in a compiler warning, some will warn of a missing return statement.  10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **return-implicit** | Fully checked |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: MSC52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmsc52cpp.html) | Checks for missing return statements (rule partially covered) |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **LANG.STRUCT.MRS LANG.STRUCT.NVNR** | Missing return statement Non-void noreturn, |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **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 |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Detect errors when converting a string to a number** |
| --- | --- | --- |
| Error Handling | [ERR-001-CPP] | When converting input from string to numeric value, there may be times where the input cannot be converted resulting in unexpected/incorrect values. |

| **Noncompliant Code** |
| --- |
| In this example, both values are numeric values converted from input. But if the input cannot be converted to a value that can be held in int, the result of the values may be unexpected. |
| #include <iostream>    void f() {  int i, j;  std::cin >> i >> j;  // ...  } |

| **Compliant Code** |
| --- |
| In this example, both inputs are checked to see if the value was properly converted and clears the input if not. |
| #include <iostream>  #include <limits>    void f() {  int i;  std::cin >> i;  if (std::cin.fail()) {  // Handle failure to convert the value.  std::cin.clear();  std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), ' ');  }    int j;  std::cin >> j;  if (std::cin.fail()) {  std::cin.clear();  std::cin.ignore(std::numeric\_limits<std::streamsize>::max(), ' ');  }    // ...  } |

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

| **Principles(s):** 10: Adopt a Secure Coding Standard – This standard is here to help reduce the amount of unlikely errors and guide developers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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) | 8.1p0 | **BADFUNC.ATOF BADFUNC.ATOI BADFUNC.ATOL BADFUNC.ATOLL** | Use of atof Use of atoi Use of atol Use of atoll |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR62-a** | The library functions atof, atoi and atol from library stdlib.h shall not be used |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR62-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr62cpp.html) | Checks for unvalidated string-to-number conversion (rule fully covered) |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

Because security is in the forefront of the mind as we develop, automation should be added both to the verify and test phase and to the monitor and detect phases. The verify and test phase is already used for testing and security checking so it makes sense to me that adding additional automated checks there to enforce the standards. Additionally, the monitor and detect phase is used to log information and alert of vulnerabilities, therefore, using said phase for automation to monitor if standards have fallen due to updates seems beneficial.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| DAT-001-CPP | Low | Likely | Medium | Low | 1 |
| DAV-001-CPP | Low | Likely | Medium | Medium | 2 |
| STC-001-CPP | High | Likely | Medium | High | 5 |
| SQL-001-CPP | High | Likely | Medium | High | 5 |
| MEM-001-CPP | High | Likely | Medium | High | 5 |
| AST-001-CPP | Low | Unlikely | Medium | Low | 2 |
| EXC-001-CPP | Low | Probable | Medium | Low | 2 |
| MIS-001-CPP | Medium | Unlikely | Low | Low | 2 |
| MIS-002-CPP | Medium | Probable | Medium | Medium | 3 |
| ERR-001-CPP | High | Likely | Medium | High | 5 |
| DAT-001-CPP | High | Likely | Medium | High | 5 |
| DAV-001-CPP | High | Likely | Medium | High | 5 |
| STC-001-CPP | Medium | Unlikely | 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 at rest | The practice of protecting stored data on devices such hard drives or in the cloud by encoding it with an encryption algorithm. This is done by encrypting the data when it is stored and needing a decryption key to restore the data to its original state when it is retrieved. This is used to protect sensitive information such as personal data and payment information that must be saved. |
| Encryption in flight | The practice of protecting data while it is being transferred from one location to another. This is done by encrypting and locking the data as is sent to another location. Once the data reaches its destination, the data needs a decryption key to access and read the data. This is used to protect data from potential attackers and interceptors in situations such as sending login credentials or personal information. |
| Encryption in use | Also known as runtime encryption, this is the practice of encrypting data that is actively being used. This is done by loading the data to the CPU as it is entered, then using encryption methods stored on the CPU to encrypt and decrypt the data in real time. This is used in situations where sensitive data is moved often within a system or processed by a third-party. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The part of the framework that involves user logins. Here users provide credentials to prove that they are who they say they are. Once the credentials are entered, they are compared against a database to ensure the credentials align. |
| Authorization | After authentication the user is granted a set of privileges in the system based on their authorization. At the higher levels of authorization, users may be able to alter the privileges granted to other users. This is in place to ensure that users are allowed to do what they need and nothing more. |
| Accounting | This part of the framework is used to monitor the activity of users while they are active within a system. Here information is logged to track things such as which files users accessed, how long they accessed the system, and which parts of the system they accessed. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 11/17/2024 | Coding standards | Zaccheus Everett | [Insert text.] |
| 1.2 | 12/08/2024 | Complete Document | Zaccheus Everett | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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