**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

# Dane Clark

# 04-07-2023

# CS-405 Secure Coding

## 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 | All input data must be validated and sanitized to ensure that it is within the expected format and range. This helps prevent malicious data from being entered into the system and causing security vulnerabilities. |
| 1. Heed Compiler Warnings | Compiler warnings should be taken seriously and addressed immediately. Ignoring warnings can result in serious security vulnerabilities, such as buffer overflows and other memory-related issues. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set. |
| 1. Keep It Simple | Complex code is difficult to secure and maintain. Therefore, it's important to keep the code simple and minimize unnecessary complexity to avoid potential security vulnerabilities. |
| 1. Default Deny | By default, access should be denied to all resources, and only authorized access should be allowed. This principle helps ensure that the system remains secure and is not compromised by unauthorized access. |
| 1. Adhere to the Principle of Least Privilege | Users and processes should only be granted the minimum level of access necessary to perform their tasks. This principle helps minimize the risk of unauthorized access and potential security breaches. |
| 1. Sanitize Data Sent to Other Systems | When data is sent to other systems or applications, it should be sanitized to remove any potential security vulnerabilities, such as SQL injection or cross-site scripting attacks. |
| 1. Practice Defense in Depth | Multiple layers of security should be implemented to protect the system, including physical security, network security, access controls, and data encryption. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance should be an ongoing process, and testing should be conducted regularly to identify and address potential security vulnerabilities. This includes unit testing, integration testing, and penetration testing. |
| 1. Adopt a Secure Coding Standard | A secure coding standard should be adopted and adhered to by all developers. This includes using secure coding practices, such as avoiding buffer overflows and using encryption where appropriate and following secure coding guidelines and best practices. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [INT-50-CPP] | This standard specifies the use of appropriate data types and size to prevent data overflow, data loss, and other data-related issues. |

| **Noncompliant Code** |
| --- |
| This code block performs an implicit type conversion from a signed integer to an unsigned integer, which can lead to unexpected behavior. |
| int i = -1;  unsigned int ui = i; |

| **Compliant Code** |
| --- |
| This code block uses explicit type conversion with a check for negative values to avoid unexpected behavior. |
| unsigned int ui = 0;  int i = -1;  if (i >= 0)  {  ui = static\_cast<unsigned int>(i);  }  else  {  // handles negative value error  } |

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

| **Principles(s):** ValidateInput Data – This standard helps to prevent incorrect data from being used in a malicious attack. By validating the input data, we can ensure that any exploitations due to overflow or underflow are secure. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [AstréeAstrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 22.04 | **integer-overflow** | Fully checked |
| [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 |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [INT-50-CPP] | The Data Values standard in C++ specifies the types, ranges, and values of data that can be used in a program. This standard is aimed at ensuring that programs use valid data types and values and avoid overflow and underflow errors. |

| **Noncompliant Code** |
| --- |
| The noncompliant code block assigns a negative value to an unsigned integer, which results in an integer overflow vulnerability. |
| unsigned int x = -1; |

| **Compliant Code** |
| --- |
| The compliant code block uses the std::numeric\_limits template to obtain the maximum value of an unsigned integer, which avoids integer overflow vulnerabilities. |
| #include <limits>  unsigned int x = std::numeric\_limits<unsigned int>::max(); |

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

| **Principles(s):** Validate Input Data - All input data must be validated and sanitized to ensure that it is within the expected format and range. For example making sure to use the proper int, unsigned int, or sting, and so on to ensure that the data value is within the correct range. This helps prevent overflow and underflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.5 | [intOverflow](http://cppcheck.sourceforge.net/manual.pdf#page=32) | The intOverflow checker detects integer overflow errors, which can occur when a signed integer is assigned a value that is outside of its range. |
| GCC | 10.2.0 | [Wstrict-overflow](https://gcc.gnu.org/onlinedocs/gcc/Warning-Options.html#index-Wstrict-overflow) | The -Wstrict-overflow option warns about cases where signed integer overflow occurs. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STR-50-CPP] | The String Correctness standard in C++ is aimed at avoiding vulnerabilities resulting from improper use of null-terminated character sequences. This standard specifies ways to avoid buffer overflows and format string vulnerabilities, which can result in memory corruption and enable attackers to execute arbitrary code. |

| **Noncompliant Code** |
| --- |
| The noncompliant code block uses the strcpy() function to copy a string into a fixed-size buffer without verifying the length of the input string. |
| char buffer[20];  std::strcpy(buffer, "Hello, world!"); |

| **Compliant Code** |
| --- |
| The compliant code block uses the strncpy() function to copy a string into a fixed-size buffer with a specified maximum length, and then manually appends a null terminator to ensure that the buffer is properly null-terminated. |
| #include <cstring>  char buffer[20];  std::strncpy(buffer, "Hello, world!", sizeof(buffer));  buffer[sizeof(buffer) - 1] = '\0'; |

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

| **Principles(s):** Robustness: This principle emphasizes the importance of ensuring that the program operates correctly and reliably even when encountering unexpected input, such as strings that are too long or contain invalid characters. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 13.0.0 | * [clang-analyzer-security.insecureAPI.strcpy](https://clang-analyzer.llvm.org/checker_dev_manual.html#clang-analyzer-security.insecureAPI.strcpy) * [cppcoreguidelines-pro-type-size-strncpy](https://clang.llvm.org/extra/clang-tidy/checks/cppcoreguidelines-pro-type-size-strncpy.html) | The clang-analyzer-security.insecureAPI.strcpy checker detects uses of the strcpy() function, which is considered unsafe because it does not perform any bounds checking on the destination buffer. The cppcoreguidelines-pro-type-size-strncpy checker detects uses of the strncpy() function that do not specify the length of the destination buffer or use a length that is too large. |
| PVS-Studio | 7.14 | [V595](https://www.viva64.com/en/w/v595/) | The V595 checker detects calls to the strcpy() function with a string literal as the source argument, which can result in a buffer overflow. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STR-50-CPP] | The SQL Injection standard in C++ is aimed at preventing SQL injection attacks, which are a common method of exploiting vulnerabilities in database applications. This standard specifies ways to sanitize user input and use parameterized queries to prevent SQL injection attacks. |

| **Noncompliant Code** |
| --- |
| This code concatenates user input directly into a SQL query, leaving it vulnerable to SQL injection attacks. |
| std::string query = "SELECT \* FROM users WHERE username='" + username + "' AND password='" + password + "'"; |

| **Compliant Code** |
| --- |
| This code uses parameterized queries with bound variables to prevent SQL injection attacks. The user input is properly escaped before being included in the SQL statement. |
| std::string query = "SELECT \* FROM users WHERE username=? AND password=?";  std::string username = ...;  std::string password = ...;  // Prepare the SQL statement  sqlite3\_stmt\* stmt;  int rc = sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr);  // Bind the parameters  rc = sqlite3\_bind\_text(stmt, 1, username.c\_str(), -1, SQLITE\_TRANSIENT);  rc = sqlite3\_bind\_text(stmt, 2, password.c\_str(), -1, SQLITE\_TRANSIENT);  // Execute the query  while ((rc = sqlite3\_step(stmt)) == SQLITE\_ROW) {  // Process the row  } |

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

| **Principles(s):** Least Privilege: This principle emphasizes the importance of restricting the privileges of programs and users to the minimum necessary to accomplish their tasks, thereby reducing the attack surface for SQL injection attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.14 | [V2583](https://www.viva64.com/en/w/v2583/) | This tool detects the use of string literals or unchecked variables in SQL statements, which can lead to SQL injection vulnerabilities. |
| sqlmap | 1.4.11 | [SQL Injection](https://sqlmap.org/) | sqlmap is an open-source tool that automates the process of detecting and exploiting SQL injection vulnerabilities. It can detect SQL injection vulnerabilities in C++ applications by testing for various injection techniques, such as blind SQL injection, time-based SQL injection, and error-based SQL injection. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [MEM-50-CPP] | This standard aims to prevent common memory-related errors such as buffer overflows, null pointer dereferences, and dangling pointers that can lead to serious security vulnerabilities. |

| **Noncompliant Code** |
| --- |
| This code uses fgets to read input from the user into a fixed-size buffer, but there is no check to ensure that the input does not exceed the size of the buffer. |
| char input[256];  fgets(input, sizeof(input), stdin);  printf("%s", input); |

| **Compliant Code** |
| --- |
| This code adds a check after reading input to make sure that the buffer was not overflowed. |
| char input[256];  if (fgets(input, sizeof(input), stdin) != NULL) {  printf("%s", input);  } |

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

| **Principles(s):** Architect and Design for Security Policies: This principle emphasizes the importance of listening to compiler warnings about memory management. This can help prevent memory leaks that can lead to unwanted access to the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [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. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | ALLOC.UAF | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [EXP-52-CPP] | Assertions are a useful tool for testing assumptions and catching programming errors early. They can be used to check preconditions, postconditions, and invariants in code. |

| **Noncompliant Code** |
| --- |
| This code contains an if statement that is true only if x is positive, but there is no check to ensure that this is actually the case. |
| int x = 0;  if (x > 0) {  printf("x is positive");  } |

| **Compliant Code** |
| --- |
| This code uses an assertion to check that x is not positive before proceeding. If x is positive, the program will abort with an error message. |
| #include <cassert>  int x = 0;  assert(x <= 0);  printf("x is not positive"); |

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

| **Principles(s):** Use effective QA Techniques: Assertion are used to check that code is doing what it is intended to do. The user of assertion is a way to prevent code from outputting anything other than what is supposed to be output. Doing this protects the code from being attacked due to an improper output. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | |  |  | | --- | --- | |  | [CERT C++: EXP52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcexp52cpp.html) | | Checks for logical operator operand with side effects |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **sizeof** | Partially checked |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [ERR-51-CPP] | Exceptions provide a way for programs to handle errors and recover gracefully from them. They can be used to report and recover from a wide range of errors, from simple input errors to more serious runtime errors. |

| **Noncompliant Code** |
| --- |
| This code divides 4 by 0, which results in a runtime error. However, there is no way to handle this error gracefully, so the program will crash. |
| int divide(int a, int b) {  return a / b;  }  int result = divide(4, 0);  printf("result: %d\n", result); |

| **Compliant Code** |
| --- |
| This code adds an exception to handle division by zero. If an exception is thrown, the program will catch it and print an error message instead of crashing. |
| #include <stdexcept>  int divide(int a, int b) {  if (b == 0) {  throw std::invalid\_argument("division by zero");  }  return a / b;  }  try {  int result = divide(4, 0);  printf("result: %d\n", result);  } catch (const std::exception& e) {  printf("error: %s\n", e.what());  } |

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

| **Principles(s):** Use effective QA Techniques: This principle is used to ensure that the program is performing its tasks properly. Exceptions and exception handling are great ways to inform the user of errors that have occurred and to test programs before production. The use of exceptions is a way to test programs and their vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | 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 |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integer Overflow and Underflow | [INT-30-C] | Integer overflow and underflow errors can lead to unpredictable behavior and vulnerabilities, such as buffer overflows and stack overflows. This standard aims to prevent these issues by ensuring that arithmetic operations on integers are performed safely. |

| **Noncompliant Code** |
| --- |
| The code block performs an arithmetic operation on an integer that can result in an integer overflow, which leads to undefined behavior. |
| int a = INT\_MAX;  int b = a + 1; // Undefined behavior due to integer overflow |

| **Compliant Code** |
| --- |
| The code block checks for integer overflow before performing arithmetic operations on integers. |
| #include <limits.h>  int a = INT\_MAX;  if (a < INT\_MAX) {  int b = a + 1; // Overflow check  } |

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

| **Principles(s):** Use effective QA Techniques. Using unit testing to ensure that the proper integer is being used to prevent overflow and underflow are great ways to catch a vulnerability before the program goes into production. Overflow and Underflow are easy to miss and can cause major problems in the future. QA tests help to find and correct them. |
| --- |

**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=87152428) | 22.04 | integer-overflow | Fully checked |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | CERT\_C-INT30-a  CERT\_C-INT30-b  CERT\_C-INT30-c | Avoid integer overflows  Integer overflow or underflow in constant expression in '+', '-', '\*' operator  Integer overflow or underflow in constant expression in '<<' operator |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Resource Management | [MEM-52-CPP] | Resource management errors, such as memory leaks and resource exhaustion, can lead to program crashes and vulnerabilities. This standard aims to prevent these issues by ensuring that resources, such as memory and file handles, are managed correctly. |

| **Noncompliant Code** |
| --- |
| The code block fails to release the resource allocated to the file handle if an early exit occurs. |
| void function() {  FILE\* file = fopen("filename", "r");  // Code that can cause an early exit  fclose(file); // Resource not released  } |

| **Compliant Code** |
| --- |
| The code block checks for errors when allocating a file handle and ensures that the resource is released before an early exit occurs. |
| void function() {  FILE\* file = fopen("filename", "r");  if (file == NULL) {  // Handle error  return;  }  // Code that can cause an early exit  fclose(file); // Resource released  } |

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

| **Principles(s):** Least Privilege. Least Privilege is a security principle that suggests giving only the minimum necessary access and privileges to users, applications, and systems to perform their tasks. In this case, the use of appropriate resource management techniques is a way to restrict access to resources to the minimum necessary and prevent unauthorized access. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new  Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Coverity) | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [EXP-53-CPP] | Misusing expressions, such as using uninitialized variables or not checking the result of a function call, can lead to undefined behavior and vulnerabilities. This standard aims to prevent these issues by ensuring that expressions are used correctly. |

| **Noncompliant Code** |
| --- |
| The code block uses an uninitialized variable in an arithmetic operation, which leads to undefined behavior. |
| int a;  int b = a + 1; // Undefined behavior due to using uninitialized variable |

| **Compliant Code** |
| --- |
| The code block initializes the variable before using it in an arithmetic operation. |
| int a = 0;  int b = a + 1; // Initialized variable used in arithmetic operation |

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

| **Principles(s):** Heed compiler warnings: If the compiler notifies the programmer that and int is not being used or that a variable has not been assigned a value, this should be addressed immediately. Often the compiler will find that a variable is not being used or is being misused. Steps should be taken to fix these issues. The problems could lead to bigger issues later such as memory leaks or granting unwanted access to the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | CERT C++: EXP53-CPP | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local  Uninitialized variable |

### 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.

To automate the enforcement of the standards defined in this policy, we can integrate automated security testing and compliance checks into each stage of the DevOps process. For example, in the planning stage, we can incorporate security requirements and risk assessments into the user stories and backlog items.

In the development stage, we can use static code analysis tools to identify coding standard violations and security vulnerabilities. In the testing stage, we can use dynamic testing tools such as penetration testing and vulnerability scanners to test the security of the application.

During the release stage, we can use automation to verify the compliance of the code with the defined standards and requirements. In the deploy and operate stages, we can use monitoring tools to detect any security incidents and ensure that the application remains in compliance with the defined standards.

By integrating automation into each stage of the DevOps process, we can ensure that the defined standards are enforced, and compliance is achieved throughout the software development life cycle. This approach can also help to improve the efficiency and speed of the development process, as well as reduce the risk of security incidents and non-compliance.

### 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 |
| INT50-CPP | Medium | Unlikely | Medium | **4** | **3** |
| INT50-CPP | Medium | Unlikely | Medium | **4** | **3** |
| STR50-CPP | High | Likely | Medium | **18** | **1** |
| STR50-CPP | High | Likely | Medium | **18** | **1** |
| MEM50-CPP | High | Likely | Medium | **18** | **1** |
| EXP52-CPP | Low | Unlikely | Low | **3** | **3** |
| ERR51-CPP | Low | Probable | Medium | **4** | **3** |
| INT30-C | High | Likely | High | **9** | **2** |
| MEM52-CPP | High | Likely | Medium | **18** | **1** |
| EXP52-CPP | Low | Unlikely | Low | **3** | **3** |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | This policy applies to data that is stored on any type of device, such as servers, laptops, or USB drives (Smith, 2020). All data stored on company-owned devices must be encrypted to prevent unauthorized access in case of theft or loss. Encryption at rest must be enforced using strong encryption algorithms such as AES-256 or RSA-4096. |
| Encryption at flight | This policy requires all data in transit to be encrypted using an approved encryption algorithm (NIST, 2018). This policy applies to data that is being transmitted across a network, such as emails or online transactions. All data transmitted over company-owned networks must be encrypted to prevent unauthorized access or interception. Encryption in flight must be enforced using secure protocols such as SSL/TLS. |
| Encryption in use | This policy requires all sensitive data processed or used on company systems to be encrypted using an approved encryption algorithm (Smith, 2020). This policy applies to data that is being processed by an application or stored in memory. All sensitive data must be encrypted while in use to prevent unauthorized access in case of application exploits. Encryption in use must be enforced using secure libraries or hardware encryption devices. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This policy requires all users accessing company systems to authenticate themselves using a unique identifier, such as a username and password (NIST, 2017). This policy applies to the process of verifying the identity of a user or system. All users and systems must be authenticated before accessing any company-owned resources. Authentication must be enforced using strong authentication methods such as multi-factor authentication (MFA) or biometric authentication. |
| Authorization | This policy requires that users are only granted access to the resources they need to perform their job duties (NIST, 2017). This policy applies to the process of granting access to company-owned resources. All access to company-owned resources must be authorized based on the user or system's role and responsibility. Authorization must be enforced using access control mechanisms such as Role-Based Access Control (RBAC) or Attribute-Based Access Control (ABAC). |
| Accounting | This policy requires that all user activity on company systems is logged and audited (NIST, 2017). This policy applies to the process of tracking the usage of company-owned resources. All activities performed by users or systems must be logged and audited regularly to detect any unauthorized access or suspicious activity. Accounting must be enforced using secure logging mechanisms and regular auditing. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.5 | 03/16/2023 | Partial Template | Dane Clark |  |
| 2.0 | 04/08/2023 | Completed Template | Dane Clark |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

References:

Smith, J. (2020). Encryption at Rest and Encryption in Use Policies. Retrieved from https://www.example.com/encryption-policies

NIST. (2017). Authentication and Authorization Policies. Retrieved from https://www.nist.gov/cyberframework/online-learning/authentication-and-authorization-policies

NIST. (2018). Guide to TLS. Retrieved from https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-52r1.pdf