**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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 14](#_Toc52464064)

[Coding Standard 7 16](#_Toc52464065)

[Coding Standard 8 18](#_Toc52464066)

[Coding Standard 9 20](#_Toc52464067)

[Coding Standard 10 22](#_Toc52464068)

[Defense-in-Depth Illustration 24](#_Toc52464069)

[Project One 24](#_Toc52464070)

[1. Revise the C/C++ Standards 24](#_Toc52464071)

[2. Risk Assessment 24](#_Toc52464072)

[3. Automated Detection 24](#_Toc52464073)

[4. Automation 24](#_Toc52464074)

[5. Summary of Risk Assessments 25](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 25](#_Toc52464076)

[7. Map the Principles 26](#_Toc52464077)

[Audit Controls and Management 27](#_Toc52464078)

[Enforcement 27](#_Toc52464079)

[Exceptions Process 27](#_Toc52464080)

[Distribution 28](#_Toc52464081)

[Policy Change Control 28](#_Toc52464082)

[Policy Version History 28](#_Toc52464083)

[Appendix A Lookups 28](#_Toc52464084)

[Approved C/C++ Language Acronyms 28](#_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 is to be validated prior to accepting and processing the data. Input data is any data coming into the software from outside of the software, and poses a significant security risk. Validation of the data will include ensuring the data is of the appropriate type, size, and does not contain any malicious code. |
| 1. Heed Compiler Warnings | Compiler warnings will be addressed prior to finalizing the code. These warnings indicate a potential issue with the software. Warnings will not prevent the code from compiling, but may provide insight into where the code may not function as intended or using obsolete features, which can pose a security risk. |
| 1. Architect and Design for Security Policies | The software should be built with a security mindset from the beginning and not as an afterthought. Design decisions and the system architecture should take into consideration the security policies. |
| 1. Keep It Simple | System security should be weighed against business practices. Implemented security should be kept simple and effective. Complex security can result in non-compliance or may reduce the effectiveness of the necessary business activities. |
| 1. Default Deny | The default position for users and other systems to access the system is to deny them access. Access to the system must be explicitly granted for specific users/systems and for specific uses. This means that even if a user has access, they should only have access to the aspects of the system they need and no more. |
| 1. Adhere to the Principle of Least Privilege | Users and systems that do have access are only permitted to system resources they require for their function and no more. This minimizes the risk of a bad actor gaining access to more sensitive areas of the system. |
| 1. Sanitize Data Sent to Other Systems | Data is to be properly sanitized before being sent out of the system. This includes removing any data that is not required, ensuring the data is in the appropriate format, and appropriate encoding. |
| 1. Practice Defense in Depth | Defense in depth is the practice of using multiple redundant layers of security. No single defense system will cover all vulnerabilities, so multiple systems should be in place to ensure that vulnerabilities are defended against appropriately. |
| 1. Use Effective Quality Assurance Techniques | Effective QA techniques are focused around testing and verifying that the security is appropriate and effective. The system should not be trusted to function as intended without appropriate testing and probing for vulnerabilities. |
| 1. Adopt a Secure Coding Standard | A well-documented and enforceable standard of coding is required. This allows for multiple developers to follow the same set of rules and guidelines to meet the security requirements of the project. |

### 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** | **Data Type Aligns with Function** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Data types shall be appropriate for the intended function. If a function is expected to return a number, ensure the chosen data type can cover the full range of numbers, to the appropriate decimal. If the data type is inappropriate, this can result in errors such as overflow. |

| **Noncompliant Code** |
| --- |
| The following code uses unsigned char as the data type, despite the intended function allowing for a negative value. |
| void func(unsigned char i) {  unsigned char result = i – 10;  std::cout << result + 0;  }  char userInput = 2;  func(userInput);  /\*The output is expected to be -8, but the actual output will be 248 \*/ |

| **Compliant Code** |
| --- |
| The following code uses the appropriate char data type and will output the expected result. |
| void func(char i) {  char result = i – 10;  std::cout << result + 0;  }  char userInput = 2;  func(userInput);  /\*The output is expected to be -8, and the actual output will be -8 \*/ |

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

| **Principles(s):** 1) Validate Input Data: The input data should align with the expected data type. If the data type is int, the input should not be a string, float, or anything that is not an int.  2) Heed Compiler warnings: The compiler will catch some instances of incorrect data types. These warnings will be heeded in order to ensure the correct data type is used where appropriate.  9) Use Effective Quality Assurance Techniques: Appropriate testing shall be utilized to ensure the data type is appropriate. This can include testing that inputs are validated to align with data type and checks for overflow, underflow, and other errors that may occur. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.10 | CheckType | Opensource tool that checks for many types of errors. The CheckType looks for various issues related to data type, typically bitwise shifts and overflow issues. |
| Coverity | 2022.12 | Quality Checker | Proprietary tool for security and quality issues. Supports compliance standards. The Quality Checker covers various issues related to data type, typically overflow |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Round Numeric Values** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Many calculations make use of decimal numbers through the float data type. When performing operations on these values, ensure that the decimal is maintained and rounded as required. If the first position to be dropped is 5 or higher, round up. Otherwise round down. Care is to be taken when performing operations with less precise data types (such as int), which drop decimal values. |

| **Noncompliant Code** |
| --- |
| The following code takes an int data type, performs an operation and expects a float type as the output. However, the decimal value is lost before the type is converted. |
| void func(int i) {  float result = i /2;  std::cout << result;  }  Int userInput = 5;  func(userInput);  /\*The output is expected to be 2.5, but the actual output will be 2 \*/ |

| **Compliant Code** |
| --- |
| The following code takes the int data type and converts it to a float prior to executing the operation. |
| void func(int i) {  float result = (float)i /2;  std::cout << result;  }  Int userInput = 5;  func(userInput);  /\*The output is expected to be 2.5, and the actual output will be 2.5 \*/ |

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

| **Principles(s):** 1) Validate Input Data: The input data shall be validated to be of the anticipated value. This can include checks for length and type.  3) Sanitize Data Sent to Other Systems: The data sent to other systems must be sanitized prior to sending out to other systems. The values should be properly encoded as required, in the appropriate format, and not include information not required to be sent.  8) Practice Defense in Depth: The value of data should be verified to be safe. This can include checks for injection of malicious code or errors with the value that can result in a breach of security.  9) Use Effective Quality Assurance Techniques: Proper testing shall be used to check that values are appropriate for their intended use. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | High (P18) | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.10 | ValueFlow | Opensource tool that checks for many types of errors. The ValueFlow looks for various issues related to the flow of values, including value and type conversions. |
| Coverity | 2022.12 | Quality Checker  Security Checker | Proprietary tool for security and quality issues. Supports compliance standards. The Quality Checker covers various issues related to data type, typically overflow. The Security Checker can check data value fields for vulnerabilities to SQL injections |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Avoid String Buffer Overflow** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | When copying data to a buffer that is not large enough to hold that data, a buffer overflow will occur. This happens particularly often when manipulating strings. In order to avoid this error, ensure the destination has sufficient size to hold the character data and the null-terminator. |

| **Noncompliant Code** |
| --- |
| The following code demonstrates an ‘Off-by-One Error’. The data is copied from the source to the destination, but does not account for the null terminator. This may result in being written 1 byte past the end of the destination. |
| void func(size\_i n, char source[n], char destination[n]) {  size\_t j;    for (j = 0; source[i] && (j < n); ++j) {  desination[j] = source[j];  }  Desination [j] = ‘\0’;  } |

| **Compliant Code** |
| --- |
| The copy function has been modified to account for the null-termination character. |
| void func(size\_i n, char source[n], char destination[n]) {  size\_t j;    for (j = 0; source[i] && (j < n - 1); ++j) {  desination[j] = source[j];  }  Desination [j] = ‘\0’;  } |

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

| **Principles(s):** 1) Validate Input Data: Ensure that input data for strings is correct, such as being within the appropriate length range.  9) Use Effective Quality Assurance Techniques: Proper testing shall be utilized to ensure that strings are handled correctly. This can include verifying string inputs, checking for the injection of malicious code through a string, verifying string transformations, translations, encryption, decryption, etc. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.10 | CheckString | Opensource tool that checks for many types of errors. The CheckString looks for various issues related to strings, such as string plus char, suspicious string compare, and overlapping data for sprintf. |
| Coverity | 2022.12 | Quality Checker  Security Checker | Proprietary tool for security and quality issues. Supports compliance standards. The Quality Checker and Security Checker can look for issues related to strings, string buffers, format, printf, etc. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Inspect Input for SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | SQL Injection is one of the most common hacking techniques. This involves malicious code being placed into SQL statements. Prior to executing any database related functions utilizing input from outside the system, verify that the input is OK to run. |

| **Noncompliant Code** |
| --- |
| A common attack appends ‘OR ‘1’ = ‘1 to the query, which will always be true. This allows a bypass of the query (such as username and password). The following code does not check for this ‘ 1 = 1’ attack and runs the query using the input as-is. |
| SELECT \* FROM db\_user WHERE username='<USERNAME>' AND password='<PASSWORD>'  /\*The malicious actor adds the following input for the username, along with an arbitrary password\*/  randomUser’ OR ‘1’=’1 |

| **Compliant Code** |
| --- |
| The following code checks for the ‘;’ and ‘=’ characters in the username, and will not run the query if they are detected. |
| SELECT \* FROM db\_user WHERE username='<USERNAME>' AND password='<PASSWORD>'  /\*The malicious actor adds the following input for the username, along with an arbitrary password\*/  randomUser’ OR ‘1’=’1  /\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/  std::regex re(".\*WHERE.\*(;|=).\*", std::regex\_constants::ECMAScript);  if (std::regex\_match(sql, re)) {  std::cout << "SQL Injection Attack Detected" << std::endl;  return false;  }  else {  return true;  } |

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

| **Principles(s):** 1) Validate Data Input: Ensure input data is free from SQL injection.  3) Architect and Design for Security Policies: As this is a common problem, ensuring that security is built from the beginning to ensure there is complete coverage.  5) Default Deny: Ensure that default deny is in place in the event that this security measure is breached. This limits the impact of the breach.  6) Adhere to the Principle of Least Privilege: As with default deny, ensure that users only have access to resources they require. This can limit the impact of a breach if SQL Injection is not prevented.  8) Practice Defense in Depth: Redundant layers of security are to be in place to cover vulnerabilities. This includes checking for and preventing SQL injection, using a default deny approach, and implementing the principle of least privilege to reduce and prevent damages in the event a single security measure is breached.  9) Use Effective Quality Assurance Techniques: Testing should be implemented to check for this type of attack where appropriate and that effective actions are taken when it is detected. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2022.12 | Security Checker | Proprietary tool for security and quality issues. Supports compliance standards. The Security Checker can look for issues related to sql injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Avoid Accessing Freed Memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Evaluating a pointer into memory that has been deallocated results in undefined behavior. It is therefore required that pointers to deallocated memory are not utilized, as this can result in vulnerabilities which may be exploited. |

| **Noncompliant Code** |
| --- |
| In this example, x is deleted and then dereferenced. |
| Struct X {  void f();  };  void g() noexcept(false) {  X \*x = new X;  delete x;  x->f();  } |

| **Compliant Code** |
| --- |
| The following code does not delete x until it is no longer required. |
| Struct X {  void f();  };  void g() noexcept(false) {  X \*x = new X;  x->f();  delete x;  } |

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

| **Principles(s):** 2) Heed Compiler Warnings: The compiler may detect a memory issue, such as a pointer to memory that has been deallocated. The warnings are to be heeded to prevent undefined behavior.  9) Use effective Quality Assurance Techniques: Ensure that appropriate tests are used to check for potential memory problems, and that memory is protected as intended. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Low | Medium (P9) | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.10 | ClassInfo  ErrorPath | Opensource tool that checks for many types of errors. The ClassInfo includes tests memory leaks in variables, structures, functions, etc. The ErrorPath includes tests for pointers, indexes, memory allocation size, etc. |
| Coverity | 2022.12 | Quality Checker | Proprietary tool for security and quality issues. Supports compliance standards. The Quality Checker includes tests for memory allocation, pointers, buffer sizes, etc. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Ensure No Side Effects in Assertions** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertions are utilized to check if a specific condition is true at a point in the program, and will terminate the program if the assertion fails. Functions should be avoided in assertion statements as they can have unexpected side effects, and assert statements should not modify values. This can be a result of ASSERT not being evaluated in the release version, only in the debug version. A VERIFY statement can be used instead, as this is evaluated in both debug and release. |

| **Noncompliant Code** |
| --- |
| The following code has the ASSERT statement modify the value of X. This means X will have different values in the debug version and in the release version. |
| ASSERT( X++ > 0); |

| **Compliant Code** |
| --- |
| The following code makes use of VERIFY instead, which will have the same result in both versions of the program. |
| VERIFY( X++ > 0); |

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

| **Principles(s):** 2) Heed Compiler Warnings: The compiler may detect an issue with the assert statements. This can include running a function or modifying a variable inside the assert statement. These warnings should be heeded.  9) Use Effective Quality Assurance Techniques: Effective testing shall be performed, which can include the use of assert or verify statements. These statements must follow the coding standard and called as appropriate to ensure the outcome of the test is accurate.  10) Adopt a secure coding standard: The use of Assert statements shall comply with the coding standard. Inappropriate use of this function may result in the code executing differently in debug and release. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.10 | CheckAssert | Opensource tool that checks for many types of errors. The CheckAssert looks for potential issues with assert statements, including side effects. |
| Coverity | 2022.12 | Quality Checker | Proprietary tool for security and quality issues. Supports compliance standards. The Quality Checker includes tests for the assert statements, including looking side effects from assert. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle All Exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | At the time an exception occurs, control is transferred to the nearest handler with a matching type. If there is no matching handler is immediately detected it will continue to search surrounding try blocks in the same thread. If no matching handler is found at all, the terminate function will be called whether or not the stack is unwound. All exceptions are to be caught and handled to ensure that the stack will be properly unwound and manage external resources prior to termination. |

| **Noncompliant Code** |
| --- |
| The following code does not catch exceptions encountered by the throwing function. As there is no handler, the exception will immediately terminate the program. |
| void func() noexcept(false);  void f() {  func();  }  int main () {  f();  } |

| **Compliant Code** |
| --- |
| The following code includes a try and catch block to handle exceptions experienced. |
| void func() noexcept(false);  void f() {  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: The warnings shall be resolved prior to finalizing the code. This may include warnings for improper try and catch blocks related to exception handling.  3) Architect and Design for Security Policies: Plan for known exceptions and ensure they are handled appropriately. If an exception cannot be handled, ensure that the code exits appropriately. This can prevent undefined behavior and security vulnerabilities.  9) Use Effective Quality Assurance Techniques: Exception handling should be verified prior to releasing the code. This ensures that exceptions are caught and handled as desired.  10) Adopt a Secure Coding Standard: All exceptions shall be handled in order to avoid potential security vulnerabilities. By following the coding standard and ensuring that exceptions are handled, this will ensure the stack will be properly unwound and external resources are managed prior to termination. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPPCheck | 2.10 | CheckExceptionSafety | Opensource tool that checks for many types of errors. The CheckExceptionSafety looks for potential issues with exception handling. |
| Coverity | 2022.12 | Quality Checker | Proprietary tool for security and quality issues. Supports compliance standards. The Quality Checker includes tests for exception handling |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do Not Modify Standard Namespaces** |
| --- | --- | --- |
| Standard Namespaces | [STD-008-CPP] | Namespaces introduce new regions for declarations and can reduce the occurrence of conflicting identifiers. Namespaces may also be extended, allowing for easy code organization. The standard name introduces the namespace std. Introducing new declarations in this namespace results in undefined behavior, which may result in vulnerabilities that could be exploited. |

| **Noncompliant Code** |
| --- |
| The following code adds a declaration to the standard namespace std, which results in undefined behavior. |
| namespace std {  char x;  } |

| **Compliant Code** |
| --- |
| The following code allows for the declaration, but it is performed in namespace that does not have a reserved name. |
| namespace notstd {  char x;  } |

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

| **Principles(s):** 2) Heed Compiler Warnings: The compiler may warn if a standard namespace is modified.  9) Use Effective Quality Assurance Techniques: Static testing of the code should indicate an issue if a standard namespace is modified.  10) Adopt a Secure Coding Standard: By following the rule to not modify a standard namespace, undefined behavior can be avoided. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 10.0.0 | CERT, DCL58-CPP | SonarQube is an open-source platform for continuous inspection of code. This checks for compliance with coding standards, which includes the rule not to modify standard namespaces. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Avoid Information Leakage Across Trust Boundaries** |
| --- | --- | --- |
| Data Protection | [STD-009-CPP] | The C++ Standard specifies:  “Nonstatic data members of a (non-union) class with the same access control are allocated so that later members have higher addresses within a class object. The order of allocation of non-static data members with different access control is unspecified. Implementation alignment requirements might cause two adjacent members not to be allocated immediately after each other; so might requirements for space for managing virtual functions and virtual base classes. Allocation of bit-fields within a class object is implementation-defined.”  “Alignment of bit-fields is implementation-defined. Bit-fields are packed into some addressable allocation unit.”  This means that padding bits may be present at any location within the class object. These padding bits contain indeterminate values, which may contain sensitive information. When passing this information across a trust boundary, it must be ensured that the padding bits of the object do not contain sensitive information. |

| **Noncompliant Code** |
| --- |
| This code copies data from arg to the userspace. Padding bits may be present to ensure proper alignment of the class data members, and this code does not account for these potential sensitive bits. |
| struct test {  int a;  char b;  char c;  };  //Copies bytes to the user space.  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);  void func(void \*usr\_buf) {  test arg{1,2,3};  copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

| **Compliant Code** |
| --- |
| This code solves the problem by serializing the structure data before copying it over. |
| struct test {  int a;  char b;  char c;  };    //Copies bytes to user space.  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void func(void \*usr\_buf) {  test arg{1, 2, 3};  unsigned char buf[sizeof(arg)];  std::size\_t offset = 0;    std::memcpy(buf + offset, &arg.a, sizeof(arg.a));  offset += sizeof(arg.a);  std::memcpy(buf + offset, &arg.b, sizeof(arg.b));  offset += sizeof(arg.b);  std::memcpy(buf + offset, &arg.c, sizeof(arg.c));  offset += sizeof(arg.c);    copy\_to\_user(usr\_buf, buf, offset);  } |

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

| **Principles(s):** 3) Architect and Design for Security Policies: By planning for security, we can ensure that data transmitted across trust boundaries is appropriate. This can include ensuring the data is the appropriate type, properly encrypted, and avoid information leakage in data padding.  7) Sanitize Data Sent to Other Systems: Data sent to other systems should not include nonrequired information. The data shall be secure and appropriate.  9) Use Effective Quality Assurance Techniques: Testing shall be performed to verify what data is sent, and if unnecessary data is included. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | MISC.PADDING.POTB | Checks padding passed along a trust boundry. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Guarantee Library Functions Do Not Overflow** |
| --- | --- | --- |
| Memory Management | [STD-010-CPP] | When data is copied into a container that is not large enough to hold it, a buffer overflow results. Therefore, copied data must align with the target container by verifying the data size and/or the container size. It is important to understand that this principle also applies to standard library functions such as std::memmove(), std::copy(), and std::memset(). When making use of these functions it remains vital to ensure the data being copied aligns with the target container size. |

| **Noncompliant Code** |
| --- |
| The std::copy() function does not inherently provide bound checking and can result in buffer overflow. The following code copies from source to target vector, but std::copy() does not expand the target vector, overflowing after the first element. |
| void func(const std::vector<int> &src) {  std::vector<int> target;  std::copy(src.begin(), src.end(), target.begin());  } |

| **Compliant Code** |
| --- |
| The following code performs the same action, but ensures that the target vector is the same size as the source vector. This will prevent overflow from occurring. |
| void func(const std::vector<int> &src) {  std::vector<int> target(src.size());  std::copy(src.begin(), src.end(), target.begin());  } |

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

| **Principles(s):** 1) Validate Input Data: The input data shall be validated to be of the anticipated value. This can include checks for length and type.  9) Use Effective Quality Assurance Techniques: Proper testing shall be used to check that values are appropriate for their intended use. Testing to ensure that overflow (and other errors) do not occur, even when using standard functions, will be performed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3p0 | BADFUNC.BO.\*  LANG.MEM.BO  LANG.MEM.TBA | CodeSonar is a static code analysis tool. The checkers are a collection of warning classes that report uses of library functions prone to internal buffer overflows, Buffer Overrun, and Tainted Buffer Access |
| Polyspace Bug Finder | R2023a | CERT C++: CTR52-CPP | Polyspace Bug Finder identifies run-time errors, concurrency issues, security vulnerabilities, and other defects in C and C++ embedded software. The checker follows coding standards, and this checks for library functions overflowing sequence container (rule partially covered). |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

Tools for automation will be considered as a part of the pre-production phase as a part of the assess and plan steps. The tools will be implemented prior to the build and test phases, and it is therefore important to ensure that appropriate tools are selected. The design phase will include the initial implementation of the tools, ensuring they are in compliance with our test driven design and secure coding policy.

The automation tools will be put to work during the build phases, as an integration into the IDE. This will allow the software to provide notification to the developer when they are not complying with our secure coding practices in real time. Verify and Test will include use of our static code analysis prior to functional testing. This is a more thorough testing of compliance with standards and security. The tools we use for building and testing will be monitored and maintained along with the code itself. This ensures that our tools are up to date and reflect any changes in our secure coding policy and any new identified vulnerabilities and threats.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Likely | Medium | High (P18) | L1 |
| STD-002-CPP | High | Probable | High | High (P18) | L1 |
| STD-003-CPP | High | Likely | Medium | High (P18) | L1 |
| STD-004-CPP | High | Likely | Medium | High (P18) | L1 |
| STD-005-CPP | High | Unlikely | Low | Medium (P9) | L2 |
| STD-006-CPP | Medium | Unlikely | Medium | Low (P4) | L3 |
| STD-007-CPP | Low | Probable | Medium | Low (P4) | L3 |
| STD-008-CPP | High | Unlikely | Medium | Medium (P6) | L2 |
| STD-009-CPP | Low | Unlikely | High | Low (P1) | L3 |
| STD-010-CPP | High | Likely | Medium | High (P18) | L1 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is the protection (encryption) of data that is stored. This may be on a disk, solid state drive, or backup media. This helps ensure that if the data is accessed by an unauthorized user, it cannot be read directly. This helps the policies of default deny, principle of least privilege, and defense in depth. Only users with the encryption key can make use of this information. |
| Encryption at flight | Encryption in flight is the protection (encryption) of data as it is being transmitted outside of the trusted boundary of the system. This ensures that the data remains secure, even if it were to be intercepted by an unauthorized user. This helps the policies of default deny, defense in depth, and the sanitization of data sent to other systems. Only the intended recipient with the appropriate encryption key can make use of this data upon receipt. |
| Encryption in use | Encryption in use is the protection (encryption) of data being access and used within the trusted boundary of the system. This ensures that the data is not decrypted while in use, preventing an unauthorized user from accessing the decrypted data at this stage (where it has traditionally been the most vulnerable). This helps the policies of default deny, principle of least privilege, and defense in depth. The system can make use of the data without decrypting it. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of validating the identity of the user. User identity should be established prior to the distribution of user login information and access. Users typically use their login credentials to verify their identity. This may include multifactor authentication for additional security. |
| Authorization | Authorization is the process of validating the user’s level of access. Their access level will determine what actions they can perform on the system, and what data they have access to. A denial of access shall be the default stance for security, which high access levels being granted more access to the system. This is opposed to defaulting to full access and restricting access at lower access levels. |
| Accounting | Accounting is the process of auditing users on the system. This can include monitoring their activity (actions performed, timestamps, data accessed, data changed, etc) and access levels as they are on the system, and ensuring that users are added and removed from the system as appropriate. Accounting can help ensure that authorization is properly setup, such as detecting when a user has access to actions or data that they were not intended to have access to. It is also important to purge user’s access from the system when they no longer require it. |

**\***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 | 11-Jun-2023 | Initial Template | Thomas Brown |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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