**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 | This principle is used to ensure validation of input. Validating input shows the importance of clearing input data to prevent potential security vulnerabilities like injection attacks. Validating user inputs ensures that the input of formats and ranges are as expected. Risks can be mitigated that are associated with malicious input. |
| 1. Heed Compiler Warnings | This principle suggests that developers should pay attention to the compiler warnings that can often alert them to potential coding mistakes and vulnerabilities that need to be addressed to ensure that the security of the software is sound. |
| 1. Architect and Design for Security Policies | This principle is the practice of integrating security policies and requirements from the start of the software development lifecycle to help developers create more resilient and secure systems. |
| 1. Keep It Simple | Keeping the development of code simple is the best way for developers to ensure that their code is understandable and easier to maintain. This can be done by avoiding unnecessary features that aren’t needed for the program to work which helps the developer mitigate unauthorized access and attacks. |
| 1. Default Deny | This principle is the idea that permissions and access is set to deny users by default. This ensures that the only users that have access are authorized to do so which will reduce the risk of attacks. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege is used to set the least number of permissions for a user that they would need to perform their tasks within the organization. |
| 1. Sanitize Data Sent to Other Systems | This principle is the practice of sanitizing or cleaning data of potentially harmful or malicious content such as cross-site scripting and injection attacks before it is sent to other systems or components. |
| 1. Practice Defense in Depth | This principle is the practice of implementing multiple layers of security such as security mechanisms, firewalls, intrusion detection systems, encryption, and access controls. With these layers of security organizations can create a stronger security posture. |
| 1. Use Effective Quality Assurance Techniques | The principle of effective quality assurance (QA) is the practice of performing tests, code reviews, and vulnerability assessments throughout the software development lifecycle. By performing these quality assurance techniques developers can identify and fix security flaws and vulnerabilities before deploying software. |
| 1. Adopt a Secure Coding Standard | This principle is to encourage developers to adopt a secure coding standard of best practices. By adopting secure coding guidelines and standards developers can ensure that they are writing secure code to mitigate common programming errors and vulnerabilities. |

### C/C++ Ten Coding Standards

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

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-C++] | Data type conventions and best practices |

| **Noncompliant Code** |
| --- |
| Code that does not adhere to the guidelines, conventions, and best practices such as potentially compromising readability, maintainability, efficiency, and overall quality of the codebase. |
| int main() {  float result = 10 / 3;  std::cout << “Result: “ << result << std:: endl;  return 0;  } |

| **Compliant Code** |
| --- |
| Code that uses appropriate data types to represent and manipulate different kinds of data in the program. |
| int main() {  float result = 10.0 / 3.0;  std::cout << “Result: “ << result << std::endl;  return 0;  } |

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

| **Principles(s):** Explicitness and Accuracy, the principle of explicitness and accuracy encourages developers to be explicit in their code and avoid implicit conversions or assumptions that could lead to unintended behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | v7.5.0 | Buffer overflows and underflows | Identifies potential buffer overflow vulnerabilities. |
| Helix QAC | 2023.2 | CERT, CWE, CWE Top 25 | Help safeguard code from potential cyberthreats and other coding vulnerabilities. |
| Parasoft C/C++ test | 2023.1 | CERT\_C-EXP20-b, CERT\_C-INT08-b, CERT\_C-INT08-c, CERT\_C-INT08-d, CERT\_CINT31-p | Avoid signed integer overflows. Avoid value change when converting between integer types. Avoid wraparounds when performing arithmetic integer operations. Avoid value change when converting between integer types. |
| TrustInSoft Analyzer | 1.38 | Signed\_overflow | Ensures that operations on signed integers do not result in overflow. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-C++] | Data value conventions and best practices standard |

| **Noncompliant Code** |
| --- |
| Code that results in incorrect data handling, loss of precision, or compromised data integrity. With the float as an integer and the bool |
| int main() {  int age = 25;  float salary = 5000;  bool isEmployed = 1;  std::string name = “John Smith”;    std:cout << “Name: “ << name << std::endl;  std::cout << “Age: “ << age << std::endl;  std::cout << “Salary: “ << salary << std::endl;  std::cout << “Employed: “ << isEmployed << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| Code that results in the correct value being assigned to the correct data type. By making the bool true and making the float 5000.0 the correct values will be used for the data types. |
| Int main() {  Std::string name = “John Smith”;  Int age = 25;  Float salary = 5000.0;  Bool isEmployed = true;  Std::cout << “Name: “ << name << std::endl;  Std::cout << “Age: “ << age << std::endl;  Std::cout << “Salary: “ << salary << std::endl;  Std::cout << “Employed: “ << std::boolalpha << isEmployed << std:: endl;  Return 0;  } |

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

| **Principles(s):** Validate Input Data to catch the mistakes from the beginning. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | INTEGER\_OVERFLOW | Ensures that unsigned integer operations do not wrap. |
| CodeSonar | 7.4p0 | ALLOC.SIZE.ADDOFLOW, ALLOC.SIZE.IOFLOW, ALLOC.SIZE.MULOFLOW, ALLOC.SIZE.SUBUFLOW, MISC.MEM.SIZE.ADDOFLOW, MISC.MEM.SIZE.BAD, MISC.MEM.SIZE.MULOFLOW, MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size. Integer overflow of allocation size. Multiplication overflow of allocation size. Subtraction underflow of allocation size. Addition overflow of size. Unreasonable size argument. Multiplication overflow of size. Subtraction underflow of size. |
| Parasoft C/C++ test | 2023.1 | CERT\_C-INT08-b, CERT\_C-INT08-c, CERT\_C-INT08-d, CERT\_CINT31-p | Avoid signed integer overflows. Avoid value change when converting between integer types. Avoid wraparounds when performing arithmetic integer operations. Avoid value change when converting between integer types. |
| Polyspace Bug Finder | R2023a | CERT C: Rule INT30-C | Checks for:  Unsigned integer overflow  Unsigned integer constant overflow  Rule partially covered. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-C++] | String handling and validation standard |

| **Noncompliant Code** |
| --- |
| This code tries to concatenate a string literal with a char and an integer which will lead to a compilation error. |
| int main() {  char\* name = “John Smith”;  int age = 25;  char\* result = “Name: “ + name + “, Age: “ + age;  cout << result << endl;  return 0;  } |

| **Compliant Code** |
| --- |
| By using the to\_string function age is converted to a string before it is concatenated. |
| int main() {  string name = “John Smith”;  int age = 25;  string result = “Name: “ + name + “, Age: “ + to\_string(age);  cout << result << endl;    return 0;  } |

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

| **Principles(s):** Validate input with appropriate string function |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | [Insert text.] | LANG.MEM.BO, LANG.MEM.TO, MISC.MEM.NTERM, BADFUNC.BO.\* | Buffer overrun, type overrun, No space for null terminator, A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Coverity | 2017.07 | STRING\_OVERFLOW, BUFFER\_SIZE, OVERRUN, STRING\_SIZE | Fully implemented |
| Parasoft C/C++ test | 2021.1 | CERT\_C-STR31-a, CERT\_C-STR31-b, CERT\_C-STR31-c, CERT\_C-STR31-d, CERT\_C-STR31-e | Avoid accessing arrays out of bounds. Avoid overflow when writing to buffer. Prevent buffer overflows from tainted data. Avoid buffer write overflow from tainted data. Avoid using unsafe string functions which may cause buffer overflows. |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Exhaustively verified |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C++] | Secure coding standard for SQL Injection prevention |

| **Noncompliant Code** |
| --- |
| The code concatenates input into the SQL query without validating input. |
| int main() {  string username;  cout << “Enter username: “;  getline(cin, username);  string query = “SELECT \* FROM users WHERE username = ‘ “ + username + “ ‘;”;    return 0;  } |

| **Compliant Code** |
| --- |
| This code will separate user input from the SQL code mitigating the risk of SQL injections. |
| int main() {  string username;  cout << “Enter username: “;  getline(cin, username);  stringstream queryStream;  queryStream << “SELECT \* FROM users WHERE username = ?”;  string query = queryStream.str();  return 0;  } |

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

| **Principles(s):** Input validation, ensure to use query parameterization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | STRING\_OVERFLOW, BUFFER\_SIZE, OVERRUN, STRING\_SIZE | Fully implemented |
| CodeSonar | 6.1p0 | LANG.MEM.BO, LANG.MEM.TO, MISC.MEM.NTERM, BADFUNC.BO.\* | BufferOverrun, Type overrun, No space for null terminator, A collection of warning classes that report uses of library functions prone to internal buffer overflows. |
| Parasoft C/C++ test | 2021.1 | CERT\_C-STR31-a, CERT\_C-STR31-b, CERT\_C-STR31-c, CERT\_C-STR31-d, CERT\_C-STR31-e | Avoid accessing arrays out of bounds. Avoid overflow when writing to buffer. Prevent buffer overflows from tainted data. Avoid buffer write overflow from tainted data. Avoid using unsafe string functions which may cause buffer overflows. |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Exhaustively verified |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C++] | Secure coding standard for memory protection |

| **Noncompliant Code** |
| --- |
| The code uses a raw pointer to allocate memory for an array but fails to deallocate the memory. |
| int main() {  int size;  cout << “Enter the size of the array: “;  cin >> size;  int\* array = new int[size];  delete[] array;  return 0;  } |

| **Compliant Code** |
| --- |
| This code uses the smart pointer unique\_ptr which handles the memory allocation and deallocation. |
| int main() {  int size;  cout << “Enter the size of the array: “;  cin >> size;  unique\_ptr<int[]> array(new int[size]);  return 0;  } |

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

| **Principles(s):** Architect for security involves training the team in writing secure memory allocation routines. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled. |
| Parasoft C/C++ test | 2021.1 | 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 | R2021a | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| Helix QAC | 2021.2 | C++3225, C++3226, C++3227, C++3229 | Declare objects with appropriate storage durations. Do not depend on the order of evaluation of subexpressions or the order in which side effects take place. Const-qualify immutable objects. Detect and remove dead code. Detect and remove unused values |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C++] | Secure coding standard for assertions |

| **Noncompliant Code** |
| --- |
| Assertions are used for debugging and this code uses the assertion to try and validate code. |
| int main() {  int value = -5;  assert(value > 0);  return 0;  } |

| **Compliant Code** |
| --- |
| The if statement is used to check if the integer is positive performing the correct error handling. |
| int main() {  int value = -5;  if (value <= 0) {  std::cerr << “Error: Value must be positive” << std::endl;  return 1;  }  return 0;  } |

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

| **Principles(s):** Architect for security involves training the team in writing secure memory allocation routines. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect. |
| CodeSonar | 6.1p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Parasoft C/C++ test | 2023.1 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-C++] | Secure coding standard for exceptions |

| **Noncompliant Code** |
| --- |
| This code block has no exception handling allowing the division operation to perform on 0 which will result in failed code and possibly a crash in the program. |
| void divide (int numerator, int denominator) {  int result = numerator / denominator;  std::cout << “Result: “ << result << std::endl;  }  int main() {  int numerator = 10;  int denominator = 0;  divide(numerator, denominator);  return 0;  } |

| **Compliant Code** |
| --- |
| This code performs error handling by checking for the division by 0 and will throw a runtime error if division by 0 is found. The try catch block will catch the exception and use error handling and feedback for the user. |
| Void divide(int numerator, int denominator) {  If (denominator == 0) {  Throw std::runtime\_error(“Division by zero”);  }  Int main() {  Int numerator = 10;  Int denominator = 0;  Try {  Divide(numerator, denominator);  } catch (const std::exception& e) {  Std::cerr << “Error: “ << e.what() << std::endl;  Return 1;  }  Return 0;  } |

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

| **Principles(s):** Architect for security involves training the team is using assertions in development codebase. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2023.1 | C++4035, C++4036, C++4037 | Handle all exceptions |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-ERR51-a, CERT\_CPP-ERR51-b | Always catch exceptions. Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point. |
| Polyspace Bug Finder | R2023a | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| RuleChecker | 20.10 | Main-function-catch-all, early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| File handling | [STD-008-C++] | Secure coding standard for file handling |

| **Noncompliant Code** |
| --- |
| This code uses nonsecure file handling practices. The code opens a file without checking to see if it was successful. |
| int main() {  std::ofstream file(“output.txt”);  file << “Hello, World!”;  file.close();  return 0;  } |

| **Compliant Code** |
| --- |
| This code checks to see if the file was opened before it writes to the file. |
| int main() {  std::ofstream file(“output.txt”);  if (file.is\_open()) {  file << “Hello, World!”;  file.close();  ] else {  std::cerr << “Error opening the file.” << std::endl;  return 1;  }  return 0;  } |

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

| **Principles(s):** A TOCTOU race condition is possible when two or more concurrent processes are operating on a shard file system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | IO.RACE | File system race condition |
| Coverity | 2017.07 | TOCTOU | Implemented |
| Helix QAC | 2023.2 | DF4851, DF4852, DF4853 |  |
| Klockwork | 2023.2 | SV.TOCTOU.FILE\_ACCESS |  |
| LDRA tool suite | 9.7.1 | 75 D | Partially implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_CFIo45-a | Avoid race conditions while accessing files. |
| Polyspace Bug Finder | R2023a | CERT C: Rule FIO45-C | Checks for file access between time of check and use (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Cross-Site Scripting prevention | [STD-009-C++] | Secure coding standard for cross-site scripting prevention |

| **Noncompliant Code** |
| --- |
| This code shows insecure handling of user input without prevention against cross-site scripting. |
| int main() {  std::string name;  std::cout << “Enter your name: “;  std::getline(std::cin, name);  std::cout << “Hello, “ + name + “!” << std::endl;  return 0;  } |

| **Compliant Code** |
| --- |
| This code uses output encoding to ensure that input from the user is properly encoded before including it in the output. |
| int main() {  std::string name;  std::cout << “Enter your name: “ ;  std::getline(std::cin, name);  std::cout << “Hello, “ << encodeForHTML(name) << “!” << std::endl;  return 0;  } |

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

| **Principles(s):** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | Bad-function | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ERR07 |  |
| LDRA tool suite | 9.7.1 | 44 S, 593 S, 594 S | Partially implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_C-ERR07-a, CERT\_C-ERR07-b | The library functions atof, atoi and atoll from library stdlib.h shall not be used. The standard library input/output functions shall not be used. |
| PC-lint Plus | 1.4 | 586 | Fully supported |
| RuleChecker | 23.04 | Bad-function | Fully checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Authentication and password storage | [STD-010-C++] | Secure coding standard for authentication and password storage. |

| **Noncompliant Code** |
| --- |
| This code collects the username and password from user input without security measures. |
| int main() {  std::string username, password;  std::cout << “Enter your username: “;  std::cin >> username;  std::cout << “Enter your password: “ ;  std::cin >> password;  return 0;  } |

| **Compliant Code** |
| --- |
| This code utilizes hashing for the password before storage to protect user information. |
| int main () {  std::string username, password;  std::cout << “Enter your username: “;  std::cin >> username;  std::cout << “Enter your password: “;  std::string rawPassword;  std::cin >> rawPassword;  std::string password = hashFunction(rawPassword);  std::memset(&rawPassword[0], 0, rawPassword.size());    return 0;  } |

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

| **Principles(s):** Do not hard code sensitive data in programs |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | HARDCODED.AUTH, HARDCODED.KEY, HARDCODED.SALT, MISC.PWD.PLAIN, MISC.PWD.PLAINTRAN | Hardcoded Authentication, Hardcoded Crypto Key, Hardcoded Crypto Salt, Plaintext Storage of Password, Plaintext Transmission of Passwor. |
| PC-lint Plus | 1.4 | 586 | Partially support |
| Polyspace Bug Finder | R2023a | CERT C: Rec. MSC18-C | Checks for: Constant or predictable block cipher initialization vector. Constant or predictable cipher key. Sensitive heap memory not cleared before release. Uncleared sensitive data in stack. Unsafe standard encryption function.  Rec. 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.

Automation should be considered during the design phase. Continuous code analysis, such as static code analysis, will scan the source code continuously to identify security vulnerabilities, coding errors, and adherence to coding standards. Automated testing can be used to validate the application’s resilience to common security threats. Automated security scanning tools will be integrated into the continuous integration and continuous deployment (CI/CD) pipeline. Automation tools will continuously monitor system behavior, collecting and analyzing security-related logs. Incident response workflows will be automated to enable swift and effective responses to security events. Automation can be used to generate compliance reports regularly, demonstrating adherence to security standards and the policy’s guidelines.

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

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | A data security measure that involves encrypting sensitive information when it is stored or at rest in databases, file systems, or any other storage media. When data is at rest, it is in a vulnerable state, susceptible to unauthorized access if someone gains physical access to the storage medium or breaches the storage system’s security. |
| Encryption at flight | A data security measure that involves encrypting sensitive information while it is being transmitted between systems or over communication channels. When data is in transit, such as when it travels over the internet or internal networks, it is exposed to potential interception by malicious actors. |
| Encryption in use | A data security measure that protects sensitive information while it is actively being used or processed within an application or system. It employs cryptographic techniques to protect data while it is being operated on by applications or processed by the system. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is a process of verifying the identity of individuals or entities attempting to access a system, network, or application. The system validates the credentials to confirm the identity of the user or entity seeking to access. |
| Authorization | This follows the process of authentication and determines what actions and privileges authenticated users or entities are granted within a system or application. When a user is successfully authenticated, the authorization process evaluates their identity against predefined access rules and permissions. |
| Accounting | This is the practice of monitoring and recording user activities, access attempts, and system events within an information system. This involves the continuous logging and tracking of various events related to system access, resource usage, and security-related incidents. |

**\***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 maps to Practice Defense in Depth
* Firewall logs maps to Adopt a Secure Coding Standard and Practice Defense in Depth
* Anti-malware logs maps to Adopt a Secure Coding Standard
* Data Type maps to Sanitize Data Sent to Other Systems
* Data Values maps to Validate Input Data
* String Correctness maps to Sanitize Data Sent to Other Systems
* SQL Injection maps to Practice Defense in Depth and Sanitize Data Sent to Other Systems
* Memory Protection maps to Architect and Design for Security Policies
* Assertions maps to Practice Defense in Depth and Adopt a Secure Coding Standard
* Exceptions maps to Practice Defense in Depth and Adopt a Secure Coding Standard
* File Handling maps to Adhere to the Principle of Least Privilege
* Cross-Site Scripting prevention maps to Sanitize Data Sent to Other Systems
* Authentication and password storage maps to Adhere to the Principle of Least Privilege

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 | 08/05/2023 | Adding features to Risk Assessment, Automated Detection, Automation, Summary of Risk Assessments, Map the principles | Drew Shepard | Professor Prasad |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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