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



# Green Pace Secure Development Policy

## Contents

[Overview](#bookmark) 2

[Purpose](#bookmark1) 2

[Scope](#bookmark2) 2

[Module Three Milestone](#bookmark3) 2

[Ten Core Security Principles](#bookmark4) 2

[C/C++ Ten Coding Standards](#bookmark5) 3

[Coding Standard 1](#bookmark6) 4

[Coding Standard 2](#bookmark7) 5

[Coding Standard 3](#bookmark8) 6

[Coding Standard 4](#bookmark9) 7

[Coding Standard 5](#bookmark10) 8

[Coding Standard 6](#bookmark11) 9

[Coding Standard 7](#bookmark12) 10

[Coding Standard 8](#bookmark13) 11

[Coding Standard 9](#bookmark14) 13

[Coding Standard 10](#bookmark15) 14

[Defense-in-Depth Illustration](#bookmark16) 15

[Project One](#bookmark17) 15

[1. Revise the C/C++ Standards](#bookmark18) 15

[2. Risk Assessment](#bookmark19) 15

[3. Automated Detection](#bookmark20) 15

[4. Automation](#bookmark21) 15

[5. Summary of Risk Assessments](#bookmark22) 16

[6. Create Policies for Encryption and Triple A](#bookmark23) 16

[7. Map the Principles](#bookmark24) 17

[Audit Controls and Management](#bookmark25) 18

[Enforcement](#bookmark26) 18

[Exceptions Process](#bookmark27) 18

[Distribution](#bookmark28) 19

[Policy Change Control](#bookmark29) 19

[Policy Version History](#bookmark30) 19

[Appendix A Lookups](#bookmark31) 19

[Approved C/C++ Language Acronyms](#bookmark32) 19

## 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 | Always check the data your system receives to make sure it’s in the right format and safe to use. This helps prevent attacks where bad data can break or take over your system. By validating input, you ensure that your system only processes good, expected data. |
| 1. Heed Compiler Warnings | Pay attention to the warnings your code compiler gives you. These warnings often point out potential issues that could become security vulnerabilities. Fixing these early helps keep your code safe and robust. |
| 1. Architect and Design for Security Policies | Plan for protection from the very beginning of your system’s design. Build in security measures as fundamental parts of your architecture. This proactive approach makes your system more secure and easier to manage. |
| 1. Keep It Simple | Simpler designs are easier to understand, maintain, and secure. Complex systems often hide bugs and vulnerabilities. By keeping your code and systems straightforward, you reduce the risk of mistakes and security issues. |
| 1. Default Deny | Start by denying all access and then allow only what is necessary. This minimizes the number of ways an attacker can exploit your system. By restricting access by default, you control what can and can’t be done, improving security. |
| 1. Adhere to the Principle of Least Privilege | Only give systems and users minimum access they to get their tasks done. This reduces the risks and negative effects associated with an account being compromised. Regularly review and adjust access levels to keep them appropriate. |
| 1. Sanitize Data Sent to Other Systems | Clean and validate data to ensure safety prior to being dispatched to other systems, preventing it from harmful content or exploiting vulnerabilities. Sanitizing data helps maintain the integrity and security of all connected systems. |
| 1. Practice Defense in Depth | Security measures should consist of several layers for system protection. In case a layer fails, others still provide protection. This layered approach makes it harder for attackers to breach your systems completely. |
| 1. Use Effective Quality Assurance Techniques | Regularly test your code and systems to find and fix security issues early. This includes code reviews, automated tests, and security assessments. Thorough testing ensures your system is secure and works as expected. |
| 1. Adopt a Secure Coding Standard | Adhere to established guidelines and ensure to use best practices when creating a safe and secure code. These standards help you avoid common security mistakes. Consistently applying secure coding standards makes your software more resilient to attacks. |

### 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** | DTP-010-CPP | Ensure Proper Data Types: Using the right type of data stops errors and makes the code clearer and safer. |

| **Noncompliant Code** |
| --- |
| Incorrectly assigning a decimal number (floating-point value) to an integer can cause loss of data. |
| void setWeight(int weight) {  weight = 120.25; // Using a floating-point value to an integer variable  } |

| **Compliant Code** |
| --- |
| Correctly using the data type for the value prevents data loss and ensures proper handling. |
| void setWeight(float weight) {  weight = 120.25; // Using a floating-point value to a floating-point variable  } |

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

|  |
| --- |
| **Principles(s):**  2. Heed compiler warnings: Using the right data types helps avoid many compiler warnings and makes the code safer.  4. Keep it simple: Clean and well-defined data types should be used to keep the code easy to understand and manage. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | [LANG.CAST.COERCE](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046622) | Don’t change a value to an enum type if it doesn’t fit within the allowed options. |
| SonarQube C/C++ | 4.10 | [IncAndDecMixedWithOtherOperators](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046625) | Ensure both the functions for allocating and deallocating are in the same place. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | DTV-020-CPP | Validate Data Values: Make sure data values are within the expected range helps avoid errors and keeps the system secure. |

| **Noncompliant Code** |
| --- |
| The height value does not check whether it is within a specific height range. |
| void setHeight(int height) {  // No validation of height value  this->height = height;  } |

| **Compliant Code** |
| --- |
| The height value is checked to ensure the height is within a specific height range. It also includes an error if it falls out of range. |
| void setHeight(int height) {  if (height >= 50 || height <= 125) { // Validating height value  this->height = height;  } else {  // Handle invalid height value  std:: cerr << “Height value is out of range!” << std::endl;  }  } |

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

|  |
| --- |
| **Principles(s):**   1. Validate input data: Make sure the data values are within the expected ranges to avoid errors and keeps the system secure.   10. Adopt a secure coding standard: Following the rules for checking data values keeps the code consistent and avoids unexpected issues. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.12 | [V555](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152206) | Makes sure that converting integers doesn’t lose or change the data, keeping values accurate. |
| Clang-Tidy | 3.9 | [Cppcoreguidelines-narrowing-conversions](https://clang.llvm.org/extra/clang-tidy/checks/cppcoreguidelines/narrowing-conversions.html) | Finds cases where changing values to a smaller type might cause loss of data. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STR-030-CPP] | Ensure String is Correct: Strings should be handled correctly to help ensure potential risks of buffer overflows and other security vulnerabilities are prevented. |

| **Noncompliant Code** |
| --- |
| Incorrect string handling can cause a risk of buffer overflow. |
| char buffer[5];  strcpy(buffer, “Hello there! How are you doing?”); // Buffer overflow |

| **Compliant Code** |
| --- |
| Correct string handling can prevent buffer overflow by checking the size. |
| char buffer[5];  strcpy(buffer, “Short”, sizeof(buffer)); // Proper string handling makes it safe  buffer[4] = ‘\0’; // Null termination |

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

|  |
| --- |
| **Principles(s):**   1. Validate input data: Correctly managing and checking strings stops buffer overflow and other security vulnerabilities.   7. Sanitize data sent to other systems: Handling and sanitizing strings properly before using or sending them reduces security risks. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | [STRING\_SIZE](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152048) | Makes sure strings have enough room for their content and a null character at the end. |
| Polyspace Bug Finder | R2024a | [CERT C: Rec. STR03-C](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152399) | Makes sure strings are handled correctly so they aren’t cut off or lost. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [SQL-040-CPP] | Prevent SQL Injection: To prevent SQL injection attacks, user inputs should be properly validated and sanitized before it is used in SQL queries. |

| **Noncompliant Code** |
| --- |
| Putting user input straight into an SQL query can let bad input change the query, causing an SQL injection attack. |
| void getProductInfo(std::shared\_ptr<sql::Connection> connection, const std::string productId) {  std::shared\_ptr<sql::PreparedStatement> prepStmt(connection-  >prepareStatement("SELECT \* FROM products WHERE id = ?"));  executeQuery(query); // Directly using user input in SQL query  } |

| **Compliant Code** |
| --- |
| Using a parameterized query safely handles user input, preventing SQL injection attacks. |
| void getProductInfo(std::shared\_ptr<sql::Connection> connection, const std::string productId) {  std::shared\_ptr<sql::PreparedStatement> prepStmt(connection-  >prepareStatement("SELECT \* FROM products WHERE id = ?”));  prepStmt->setString(1, productId); // Parameterized queries used  to safely handle user input std::shared\_ptr<sql::ResultSet> resultSet(prepStmt->executeQuery());  } |

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

|  |
| --- |
| **Principles(s):**   1. Validate input data: User inputs should be checked before being used in SQL queries to help prevent SQL injection attacks.   7. Sanitize data sent to other systems: Clean inputs used in SQL queries to make sure only safe data is processed. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klocwork | 2024.2 | [CERT.DCL.STD\_NS\_MODIFIED](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046686) | Ensure that library functions don’t exceed their limits. |
| CodeSonar | 8.1p0 | [BADFUNC.BO.\*](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046690) | The code for SQL injections are analyzed for any vulnerabilities and make mitigation suggestions. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [MEM-050-CPP] | Protect Memory Usage: Managing memory correctly prevents leaks and security problems. |

| **Noncompliant Code** |
| --- |
| Memory is used but not released, causing a memory leak. |
| void createArray() {  int\* array = new int[100];  // Forgets to delete the allocated memory  } |

| **Compliant Code** |
| --- |
| Ensure that memory is allocated and released properly to avoid any memory leaks. |
| void createArray() {  int\* array = new int[100];  // Use the array  delete[] array; // The allocated memory is properly releases  } |

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

|  |
| --- |
| **Principles(s):**  8. Practice defense in depth: Adding several layers of protections makes memory management more secure against attacks.  9. Use effective quality assurance techniques: Conducting regular tests and reviewing codes help to find and fix memory problems early. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klocwork | 2024.2 | [UNINIT.HEAP.MIGHT](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046609) | Makes sure heap memory is set up before being used. |
| Polyspace Bug Finder | R2024a | [CERT C: Rec. MEM00-C](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152150) | Ensures memory is managed correctly to avoid leaks and misakes. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [AST-060-CPP] | Use Assertions Correctly: Using assertions correctly helps find bugs and ensures the code works properly. |

| **Noncompliant Code** |
| --- |
| The function sets the speed without checking if it’s in a safe range, which can cause problems. |
| void setSpeed(int speed) {  this->speed = speed; // Doesn’t check if the speed is within a specific range  } |

| **Compliant Code** |
| --- |
| An assertion checks that the speed is within a safe range, preventing mistakes and catching bugs early. |
| void setSpeed(int speed) {  assert(speed >= 0 && speed <= 100); // Checks if the speed is within a specific range  this->speed = speed;  } |

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

|  |
| --- |
| **Principles(s):**  3. Architect and design for security policies: When assertions are used, it checks that assumptions and rules are always correct.  10. Adopt a secure coding standard: Following a standard that uses assertions maintains the quality of the code and keeps it secure. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | [clang-analyzer-core.UndefinedBinaryOperatorResult](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046609) | Finds problems when comparisons are done with variables that haven’t been given a value yet. |
| Coverity | 2017.07 | [ASSERT\_SIDE\_EFFECT](http://www.apple.com) | Makes sure that assertions are used properly and don’t cause any unexpected changes. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [EXC-070-CPP] | Handle Exceptions Properly: Properly handle exceptions to ensure error management is strong and secure. |

| **Noncompliant Code** |
| --- |
| There is no proper handling of the catch-all exceptions. |
| try {  // Line of code throwing exception  } catch (…) {  // Catch all without handling  } |

| **Compliant Code** |
| --- |
| The specific exceptions are caught and taken care of properly. |
| try {  // Line of code throwing exception  } catch (…) {  std::cerr << “Error: “ << e.what() << std::end1; // Proper exception handling  } |

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

|  |
| --- |
| **Principles(s):**  3. Architect and design for security policies: Proper exception handling helps the system manage errors smoothly without sacrificing the security.  4. Keep it simple: Make exception handling straightforward so the code is easy to read, maintain, and understand. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2024a | [EXP05-C](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152191) | Prevents removing ‘const’ from variables, which helps ensure correct exception handling and avoids unpredictable issues. |
| Clang | 3.9 | [-Wexceptions](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046587) | Gives warnings if there are issues with exception handling. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Validation | [VAL-080-CPP] | Validate User Inputs: Making sure user inputs are valid prevents security problems and makes the program more reliable. |

| **Noncompliant Code** |
| --- |
| The input from the user is not validated before use. |
| std::string input =getUserInput();  // Does not include validation |

| **Compliant Code** |
| --- |
| Ensuring user input is validated before it is being used. |
| std::string input = getUserInput();  if (isValid(input)) {  // Uses validation input  } |

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

|  |
| --- |
| **Principles(s):**  1. Validate Input Data: Making sure user inputs are valid prevents security problems and makes the program more reliable.  10. Adopt a secure coding standard: Using a standard for input validation keeps the code consistent and secure. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 3.9 | [Cert-str34-c](https://clang.llvm.org/extra/clang-tidy/checks/bugprone/signed-char-misuse.html) | Makes sure input is checked to avoid common security problems. |
| Polyspace Bug Finder | R2024a | [CERT C: Rec. STR07-C](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152413) | Checks the size of input strings to stop buffer overflows and other input problems. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Secure File Handling | [SEC-090-CPP] | Handle Files Securely: Handling files correctly keeps them safe, preventing unauthorized access and data corruption. |

| **Noncompliant Code** |
| --- |
| The function tries to read a file without checking if it opened correctly, which could cause runtime errors. |
| void readSecureFile(const std::string& mySecureFile) {  std::ifstream file(mySecureFile);  std::string content;  while (file >> content) {  // Read file contents  }  // Does not check if file was opened successfully  } |

| **Compliant Code** |
| --- |
| The function ensures the file opens correctly before reading it and ensure it closes, keeping file handling safe and avoiding errors. |
| void readSecureFile(const std::string& mySecureFile) {  std::ifstream file(mySecureFile);  // ensures file is opened correctly before reading  // otherwise, error message is displayed  if (!file.is\_open) {  std::cerr << “Error: Unable to open: “ << mySecureFile << std::endl;  return;  }  std::string content;  while (file >> content) {  // Reads file contents  }  file.close(); // Ensure file is closed for safety  } |

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

|  |
| --- |
| **Principles(s):**  5. Default deny: Limiting the access to files by default will stop access that is unauthorized.  6. Adhere to the principle of least privilege: Using only the minimum permissions needed for file operations reduces security risks. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | [ALLOC.LEAK](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046824) | Make sure to close files when you’re done using them. |
| Klocwork | 2024.2 | [RH.LEAK](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046824) | Makes sure files are properly closed when no longer using. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Safe Variable Initialization | [INI-100-CPP] | Ensure Safe Initialization of Variables: Properly initializing variables ensures they don’t have random values, which can cause unpredictable behavior and security issues. |

| **Noncompliant Code** |
| --- |
| Using the variable data without initializing it can cause unpredictable behavior and security problems. |
| void processData() {  int data;  if (data > 0) {  // Process data  }  } |

| **Compliant Code** |
| --- |
| Before using it, the variable is set to a known value, ensuring it behaves as expected and avoiding security problems. |
| void processData() {  int data = 0; // Properly initializes the variable  if (data > 0) {  // Process data  }  } |

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

|  |
| --- |
| **Principles(s):**  2. Heed compiler warnings: Variables should be initialized properly to get rid of compiler warnings that are about uninitialized variables.  10. Adopt a secure coding standard: Standars should be used for initializing variables to keep the code consistent and avoi unexpected issues. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 3.9 | [cppcoreguidelines-pro-type-member-init](https://clang.llvm.org/extra/clang-tidy/checks/cppcoreguidelines/pro-type-member-init.html) | Makes sure all vairables are set to a value before being used. |
| Polyspace Bug Finder | R2024a | [CERT C: Rec. DCL01-C](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152192) | Prevents reusing variable names in different parts of the code to avoid confusion and mistakes. |

### 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 effectively automate security and compliance within Green Pace’s DevOps process, it is essential to employ automated tools in the pre-production phase to monitor threats, update compliance requirements, and prioritize tasks. Integrating security checks and best practices such as OWASP into the development pipeline is crucial. It is vital to ensure secure builds by automatically validating trusted sources and conducting automated vulnerability scans and security tests.

During the production phase, deployment management should be automated, accompanied by implementing penetration tests and proper configuration of security settings. Continuous monitoring with tools capable of analyzing logs and detecting incidents is imperative. Automating responses to security threats, including blocking attacks and restoring safe states, is essential. Regular automated checks against security baselines will help maintain system stability and ensure swift recovery from any security issues. This approach will significantly enhance both the DevOps process's security and efficiency.

### 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** |
| --- | --- | --- | --- | --- | --- |
| **DTP-010-CPP** | High | Likely | Medium | High | 1 |
| **DTV-020-CPP** | High | Likely | Medium | High | 1 |
| **STR-030-CPP** | High | Likely | Medium | High | 1 |
| **SQL-040-CPP** | High | Likely | High | High | 1 |
| **MEM-050-CPP** | High | Likely | Medium | High | 1 |
| **AST-060-CPP** | Medium | Probable | Low | Medium | 2 |
| **EXC-070-CPP** | High | Probable | Medium | High | 1 |
| **VAL-080-CPP** | High | Likely | High | High | 1 |
| **SEC-090-CPP** | High | Likely | Medium | High | 1 |
| **VAR-100-CPP** | Medium | Probable | Low | Medium | 2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | This policy encrypts stored data on physical devices, protecting it from unauthorized access and complying with regulations. |
| Encryption in flight | This policy encrypts data during transmission over networks, preventing interception and tampering, essential for securing network communications. |
| Encryption in use | This policy keeps data safe while its being used in memory, safeguarding it from unauthorized access during computation. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This policy makes sure that only approved users or devices can get into a system by verifying their identity using passwords, MFA, or certificates. |
| Authorization | This policy controls what authenticated users or devices can access and do, using role-based access to enforce permissions and protect resources. |
| Accounting | This policy logs and tracks user and device activities for monitoring, auditing, and analysis, helping to detect incidents and ensure accountability. |

**\***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 |  |
| **2.0** | 08/06/2024 | Security Policy Template | Dhevie Miller |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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