**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 | Ensuring user inputs are validated prevents malicious data from being exploiting vulnerabilities in the system/code. This ensures that only properly formatted data can be/is processed and reduces the risks of injection attacks or buffer overflows. |
| 1. Heed Compiler Warnings | Compiler warnings can point out potential issues in the code that can lead to vulnerabilities. Ensuring these warnings are addressed or acknowledged proactively help to identify bugs or insecure code writing practices early in the development process. |
| 1. Architect and Design for Security Policies | While developing and maintaining best security practices in the design phases can ensure that the system will adhere to security policies and is being built to handle potential threats. It can also be beneficial so that we do not have to retrofit to these security requirements later in the development process. |
| 1. Keep It Simple | Simple systems are (generally) easier to secure as well as maintain. By avoiding complex code, this reduces the likelihood that something we wrote can be exploited or that we have accidentally introduced vulnerabilities due to our code writing. |
| 1. Default Deny | This principle makes sure that permission are denied unless they are explicitly allowed. This ensures only authorized users can access information that they need to prevent potential breeches of private data. |
| 1. Adhere to the Principle of Least Privilege | Related to default deny, this principle ensures that users and systems should only have the permissions required to perform the required functions of their role. This reduces the risk of potential damage from compromised accounts or malicious users. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data can ensure that only intended information is sent between systems and prevents vulnerabilities such as injection attacks or cross-site scripting between connected environments. This is especially useful when interacting with third-party API’s db’s, or external systems because improperly sanitized data can compromise not only our system but others as well. |
| 1. Practice Defense in Depth | Defense in depth implements multiple layers of security and ensures a robust defense strategy. Each layer can act as a backup if another fails. This approach can include physical security, network firewalls, application protections and user authentication. By utilizing these overlapping safeguards, the system can be better protected against new threats or unforeseen vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Effective QA is more than simple testing. I can include aspects of static and dynamic code analysis, specific test cases, and injection or penetration testing. These methods allow both function bugs or security vulnerabilities are caught and fixed sooner and can lead to a more secure and reliable software on delivery. |
| 1. Adopt a Secure Coding Standard | Secure coding provides developers with an ability for writing code that actively mitigates common security vulnerabilities like buffer overflows or improper error handling. By using and enforcing these standards across development teams, companies or organizations can make sure that there is consistent priority afforded to security and fosters a culture of proactive risk management. |

### 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-DAT] | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| Using an unsigned int for negative values can result in unexpected behavior |
| // a.cpp  struct S {  int a;  };  // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| Use a signed integer to store negative values appropriately |
| struct S {  int a;  };  // a.cpp  #include "S.h"  // b.cpp  #include "S.h" |

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

| **Principles(s):** Validate Input Data: Ensures data type compatibility and correctness, preventing errors caused by mismatched or unintended definitions.  Architect and Design for Security Policies: Enforcing one-definition rules ensures consistency in design and prevents hidden vulnerabilities during compilation and linking. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.3 | C++1067, C++1509, C++1510 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| Polyspace Bug Finder | R2024a | CERT C++: DCL60-CPP | Checks for inline constraints not respected (rule partially covered). |
| Astrée | 22.10 | type-compatibility, definition-duplicate, undefined-extern, undefined-extern-pure-virtual, external-file-spreading, type-file-spreading | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-DAT] | Avoid Modifying Standard Namespaces |

| **Noncompliant Code** |
| --- |
| Modifying the std namespace by adding a declaration results in undefined behavior. |
| namespace std {  int x; // Noncompliant  } |

| **Compliant Code** |
| --- |
| Instead of modifying the std namespace, use a user-defined namespace to avoid undefined behavior. |
| namespace nonstd {  int x; // Compliant  } |

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

| **Principles(s):** Architect and Design for Security Policies: Ensures compatibility with standard libraries and prevents undefined behavior.  Adhere to the Principle of Least Privilege: Reduces the risk of unintended modifications to critical namespaces. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.STRUCT.DECL.SNM | Detects modification of standard namespaces. |
| Helix QAC | 2024.3 | C++3180, C++3181, C++3182 |  |
| SonarQube C/C++ Plugin | 4.10 | S3470 |  |
| Polyspace Bug Finder | R2024a | CERT C++: DCL58-CPP | Fully covers rule compliance. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-STR] | Do Not Confuse Narrow and Wide Character Strings and Functions |

| **Noncompliant Code** |
| --- |
| Incorrect use of strncpy() with wide characters leads to potential truncation or undefined behavior. |
| #include <stddef.h>  #include <string.h>  void func(void) {  wchar\_t wide\_str1[] = L"0123456789";  wchar\_t wide\_str2[] = L"0000000000";  strncpy(wide\_str2, wide\_str1, 10); // Noncompliant  } |

| **Compliant Code** |
| --- |
| Use appropriate functions for the respective string types |
| #include <string.h>  #include <wchar.h>  void func(void) {  wchar\_t wide\_str1[] = L"0123456789";  wchar\_t wide\_str2[] = L"0000000000";  wcsncpy(wide\_str2, wide\_str1, 10); // Proper-width function for wide strings  char narrow\_str1[] = "0123456789";  char narrow\_str2[] = "0000000000";  strncpy(narrow\_str2, narrow\_str1, 10); // Proper-width function for narrow strings  } |

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

| **Principles(s):** Validate Input Data: Guarantees appropriate use of string operations, minimizing the risk of buffer overflows or truncation.  Keep It Simple: Reduces complexity by using functions specific to the data type, ensuring clarity and correctness. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | P27 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | LANG.MEM.BO, LANG.MEM.TBA | Detects buffer overruns and tainted buffer access |
| Polyspace Bug Finder | R2024a | CERT C: Rule STR38-C | Fully implemented |
| Helix QAC | 2024.3 | C0432, C++0403 | Helix QAC |
| RuleChecker | 24.04 | wide-narrow-string-cast, wide-narrow-string-cast-implicit | Partially checked |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-SQL] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| Directly concatenating user input into SQL queries can lead to injection attacks |
| string query = “SELECT \* FROM users WHERE id = “ + userInput; |

| **Compliant Code** |
| --- |
| Use parameterized queries to prevent SQL injection. |
| PreparedStatement stmt = connection.prepareStatement("SELECT \* FROM users WHERE id = ?");  stmt.setString(1, userInput); |

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

| **Principles(s):** Validate Input Data: Sanitizing inputs ensures that malicious queries cannot exploit vulnerabilities.  Sanitize Data Sent to Other Systems: Prevents untrusted data from propagating vulnerabilities to external systems or databases. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | tainted-sql-query | Detects untrusted input in SQL queries. |
| CodeSonar | 8.1p0 | LANG.MEM.TAINTED, SQL\_INJECTION | Identifies SQL injection vulnerabilities in C/C++ code. |
| Klocwork | 2024.3 | SV.SQL, SV.DATA.DB | Detects SQL injection vulnerabilities through tainted data flow. |
| SonarQube | 9.9 | S2077, S3649 | Highlights vulnerabilities in dynamic SQL queries. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-MEM] | Prevent memory leaks |

| **Noncompliant Code** |
| --- |
| Allocating memory without freeing it causes memory leaks. |
| int\* data = new int[100];  //memory not freed |

| **Compliant Code** |
| --- |
| Using delete[] can free allocated memory. |
| int\* data = new int[100];  //properly freeing data  delete[] data; |

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

| **Principles(s):** Manage Resources Efficiently: Ensures proper allocation and deallocation of memory to prevent resource exhaustion.  Practice Defense in Depth: Adds layers of safeguards by ensuring resources are explicitly managed during the program's lifecycle. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.3 | DF4761, DF4762, DF4766, DF4767 | Detects improperly managed dynamic memory. |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM53-a | Ensures malloc/realloc is not used for objects requiring constructors. |
| Polyspace Bug Finder | R2024a | CERT C++: MEM53-CPP | Fully covers checks for uninitialized objects allocated with malloc. |
| PVS-Studio | 7.33 | V630, V749 | Identifies potential misuse of dynamic memory operations. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-ASR] | Avoid Dangerous Assertions |

| **Noncompliant Code** |
| --- |
| Using std::terminate() for assertion checking does not provide debugging information. |
| #include <exception>  void checkValue(int value) {  if (value < 0) {  std::terminate(); // Noncompliant  }  } |

| **Compliant Code** |
| --- |
| Using assert provides helpful debugging information when the condition fails. |
| #include <cassert>  void checkValue(int value) {  assert(value >= 0); // Compliant  } |

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

| **Principles(s):** Validate Input Data: Assertions verify conditions during runtime, ensuring program state meets expectations.  Keep It Simple: Provides a straightforward way to enforce constraints without complex error-checking code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | P25 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.3 | MSC50-CPP | Identifies improper use of std::terminate() for assertion checking. |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MSC50-a | Detects inappropriate assertion handling using std::terminate(). |
| Polyspace Bug Finder | R2024a | CERT C++: MSC50-CPP | Fully checks for improper assertion handling. |
| PVS-Studio | 7.33 | V620 | Detects misuse of assertions or abrupt termination calls. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-EXC] | Guarantee That Container Indices and Iterators Are Within the Valid Range |

| **Noncompliant Code** |
| --- |
| Failing to catch exceptions can result in unexpected crashes |
| void throwing\_func() noexcept(false);  int main() {  throwing\_func(); // Noncompliant: Unhandled exception  } |

| **Compliant Code** |
| --- |
| Catch exceptions and handle them appropriately to ensure program stability. |
| #include <iostream>  void throwing\_func() noexcept(false);  int main() {  try {  throwing\_func();  } catch (...) {  std::cerr << "Exception caught. Exiting gracefully." << std::endl;  }  } |

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

| **Principles(s):** Practice Defense in Depth: Catching exceptions ensures fallback mechanisms and prevents abnormal program termination.  Keep It Simple: Centralized exception handling provides a clear, structured approach to error management. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | main-function-catch-all, early-catch-all | Partially checks for unhandled exceptions. |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Identifies unreachable or missing exception handlers. |
| Helix QAC | 2024.3 | C++4035, C++4036, C++4037 | Detects unhandled exceptions and improper catch constructs. |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR51-a, CERT\_CPP-ERR51-b | Ensures all thrown exceptions are caught appropriately. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-008-CTR] | Guarantee That Container Indices and Iterators Are Within the Valid Range |

| **Noncompliant Code** |
| --- |
| Accessing container elements without range checking can cause undefined behavior. |
| #include <vector>  void accessElement() {  std::vector<int> v = {1, 2, 3};  int val = v[5]; // Noncompliant: Out-of-bounds access  } |

| **Compliant Code** |
| --- |
| Always validate the index before accessing container elements. |
| #include <vector>  #include <iostream>  void accessElement() {  std::vector<int> v = {1, 2, 3};  if (5 < v.size()) {  int val = v[5]; // Compliant: Index is validated  } else {  std::cerr << "Index out of range" << std::endl;  }  } |

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

| **Principles(s):** Validate Input Data: Ensures safe access to container elements, reducing risks of undefined behavior.  Keep It Simple: Simple range validation ensures safe and predictable operations without unnecessary complexity. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P1 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2024.3 | C++1753, C++1754 | Detects unsafe container access. |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-CTR50-a | Validates container index usage. |
| Polyspace Bug Finder | R2024a | CERT C++: CTR50-CPP | Checks for out-of-bounds access to containers. |
| PVS-Studio | 7.33 | V595 | Identifies unsafe iterator and index usage. |

**Coding Standard 9**

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions and Error Handling | [STD-009-ERR] | Avoid Uncaught Exceptions in Static and Thread Storage Duration |

| **Noncompliant Code** |
| --- |
| Using a global static variable that might throw an exception during construction. |
| struct S {  S() noexcept(false); // Constructor that may throw  };  static S globalS; // Exception during construction cannot be caught |

| **Compliant Code** |
| --- |
| Use a function to manage the initialization of a static variable, allowing exceptions to be caught. |
| struct S {  S() noexcept(false);  };  S& globalS() {  try {  static S s;  return s;  } catch (...) {  // Log error or handle exception  throw; // Re-throw or terminate as necessary  }  } |

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

| **Principles(s):** Practice Defense in Depth: Wrapping static initialization in functions ensures robust error handling during program startup.  Architect and Design for Security Policies: Proper exception handling mitigates risks of critical failures during system initialization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | potentially-throwing-static-initialization | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR58 | Checks for exceptions during static initialization |
| Helix QAC | 2024.3 | C++4634, C++4636, C++4637, C++4639 | Identifies improper exception handling in static/global contexts |
| Polyspace Bug Finder | R2024a | CERT C++: ERR58-CPP | Fully checks exceptions raised during program startup |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | [STD-010-FIO] | Ensure File Resources Are Properly Closed |

| **Noncompliant Code** |
| --- |
| Failing to close a file explicitly before program termination leads to resource leakage. |
| #include <fstream>  #include <string>  #include <exception>  void processFile(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // Perform operations on the file  std::terminate(); // File not closed  } |

| **Compliant Code** |
| --- |
| Explicitly closing the file ensures resources are properly released. |
| #include <fstream>  #include <string>  #include <exception>  void processFile(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  // Perform operations on the file  file.close(); // Explicitly close the file  if (file.fail()) {  // Handle error  }  std::terminate();  } |

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

| **Principles(s):** Practice Defense in Depth: Properly managing file resources provides an additional layer of protection against resource exhaustion and ensures system stability.  Use Effective Quality Assurance Techniques: Ensuring files are closed correctly can be tested during QA to catch potential resource leaks and improve the system's reliability. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Detects file resource leaks |
| Helix QAC | 2024.3 | DF4786, DF4787, DF4788 | Identifies file resource mismanagement |
| Klocwork | 2024.3 | RH.LEAK | Detects resource leaks |
| Polyspace Bug Finder | R2024a | CERT C++: FIO51-CPP | Checks for resource leaks (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.

To integrate automation into the Green Pace DevOps process, security should be included at every stage of the DevSecOps lifecycle. During the pre-production phase, tools like SonarQube and CodeSonar can be used to automate threat analysis, check regulatory compliance, and enforce secure coding standards during design and build steps. Continuous Integration (CI) pipelines can run automated tests like static code analysis and dependency scans to validate code and ensure compliance before deployment. Tools for vulnerability scanning and penetration testing can confirm the security of each build before it moves to production.

In production, automation can help manage configurations and ensure secure deployments using tools like Docker for containerization. Real-time monitoring systems, like SIEM and intrusion detection tools, can provide continuous oversight and detect anomalies. Automated incident response tools can quickly block attacks, isolate breaches, and roll back compromised systems, reducing response time to security incidents. After deployment, baseline assessments and hardening scripts can help maintain compliance and strengthen the system against future threats. By using these automated processes, Green Pace can improve the security of its software lifecycle while reducing the need for manual work.

### 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-DAT | High | Unlikely | High | P3 | L3 |
| STD-002-DAT | High | Unlikely | Medium | P6 | L2 |
| STD-003-STR | High | Likely | Low | P27 | L1 |
| STD-004-SQL | High | Likely | Medium | P18 | L1 |
| STD-005-MEM | High | Likely | Medium | P18 | L1 |
| STD-006-ASR | Medium | Unlikely | Low | P25 | L2 |
| STD-007-EXC | Low | Probable | Medium | P4 | L3 |
| STD-008-CTR | High | Likely | Medium | P1 | L2 |
| STD-009-ERR | Low | Likely | Low | P9 | L2 |
| STD-010-FIO | Medium | Unlikely | Medium | P4 | L3 |

### 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 | Encryption at rest secures data stored on disk, such as in files, databases, or backups, by making it inaccessible to unauthorized users. This policy ensures sensitive or regulated information remains safe, even if physical storage devices are lost or stolen.  Policy:  All data stored at rest must use AES-256 encryption or an equivalent industry-standard.  Encryption keys must be securely managed using a Key Management System (KMS), and access to these keys must be limited to authorized personnel.  Backups must follow the same encryption standards as the primary data. |
| Encryption in flight | Encryption in flight secures data during transmission over networks by encrypting it before sending and decrypting it upon receipt. This policy ensures the confidentiality and integrity of data during communication, protecting it from interception or tampering.  Policy:  Data transmitted across public or internal networks must use TLS 1.2 or higher encryption.  Secure communication protocols, such as HTTPS or VPNs, must be used for all data exchanges between systems, APIs, and external endpoints.  Certificates for secure communication must come from trusted authorities and be regularly rotated to maintain security. |
| Encryption in use | Encryption in use protects data actively processed in memory from being exposed to unauthorized access. This is particularly critical in cloud or multi-tenant environments to safeguard sensitive computations.  Policy:  Mechanisms such as Intel SGX or AWS Nitro Enclaves must be used to encrypt sensitive data while it is being processed in memory.  Applications must isolate sensitive data during processing and implement safeguards against unauthorized access or memory leaks.  Developers must follow secure coding practices to ensure no sensitive information is exposed during active operations. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication verifies the identity of users or systems before granting access, ensuring only legitimate entities can access resources. Strong authentication methods minimize the risk of unauthorized access.  Policy:  Multi-factor authentication (MFA) is required for all user logins.  Password policies must enforce complexity, expiration, and uniqueness to reduce password-related vulnerabilities.  Mutual TLS or API keys must be used for authenticating service-to-service communication. |
| Authorization | Authorization defines what authenticated users or systems are allowed to access or perform, ensuring compliance with organizational policies and role-based permissions.  Policy:  Role-based access control (RBAC) must be implemented to assign permissions based on user roles and responsibilities.  The principle of least privilege must be followed, granting users and systems access only to the resources and actions necessary for their roles.  Access rights must be reviewed quarterly, and changes to access levels require manager approval and documentation. |
| Accounting | Accounting tracks and logs all user and system activities, providing visibility into operations and ensuring accountability. These logs support anomaly detection and forensic investigations.  Policy:  All user activities, including logins, database changes, and file access, must be logged and securely stored for at least 90 days.  System-level logs must include timestamps, user IDs, and details of actions taken.  Logs should be monitored continuously through a Security Information and Event Management (SIEM) system for anomaly detection and incident response. |

**\***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.

Principles:

* + 1. Least Privilege
    2. Separation of Duties
    3. Defense in Depth
    4. Fail-Safe Defaults
    5. Accountability
    6. Simplicity
    7. Open Design
    8. Layered Security
    9. Ease of Use
    10. Automation

|  |  |  |
| --- | --- | --- |
| Standard | Principles Applied | Justification |
| Authentication | 1, 5, 10 | Supports access control, user tracking, and streamlined MFA implementation. |
| Authorization | 1, 2, 8 | Limits access by role, separates critical tasks, and adds layered controls. |
| Accounting | 3, 5, 7 | Ensures logging, accountability, and transparent mechanisms for tracking actions. |
| Encryption at Rest | 3, 4, 8 | Encrypts sensitive data by default, even if other security measures fail. |
| Encryption in Flight | 3, 6, 8 | Protects data during transmission with simple and layered security protocols. |
| Encryption in Use | 3, 8, 10 | Safeguards active data through layered runtime encryption and automated processes. |

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.0.1 | 12/02/2024 | Implement security policies and encryption descriptions in line with “Project One” outline. | Joel Hays |  |
| [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 |