**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 | Validate input to ensure the data entering a system is correct and secure before processing. Can eliminate most software vulnerabilities by being skeptical of all data sources. This process of checking and sanitizing data can be an effective first line of defense against security attacks. |
| 1. Heed Compiler Warnings | Utilizes the highest warning level available for a compiler to aid in identifying security flaws. Another huge benefit to the detection happening at the compiler level is that vulnerabilities are caught very early, often before they can become actual vulnerabilities. Compilers are also designed to recognize patterns that may lead to vulnerabilities, so it makes it easier for developers to identify and ‘heed’ their warnings. |
| 1. Architect and Design for Security Policies | Throughout the process of developing software, a software architecture and design plan must be implemented with security policies in mind. This allows security considerations to be integrated from the beginning, rather than being added as an afterthought which could cause vulnerabilities. A good architecture also makes it easier to accommodate changes in security requirements, which can arise from various sources such as new regulations. |
| 1. Keep It Simple | Keep the design as simple as possible. Unnecessary complexity can lead to an increase in errors and make it much more difficult to achieve a high level of security. Also, in the world of security, less is more. Fewer components and simpler interactions mean fewer opportunities for vulnerabilities to be introduced or exploited. |
| 1. Default Deny | By default, users are denied access to a system. This means that access to the system is handled on a case-by-case basis and follows a set of conditions that follow a protection scheme. This limits the opportunities for attackers to exploit unused or unnecessary entry points. |
| 1. Adhere to the Principle of Least Privilege | Similar to default deny, adhering to the principle of least privilege limits user access to avoid unnecessary security risks. This is done by giving only the bare minimum of privileges needed to users to complete their tasks. This can help system security in a myriad of ways, but one way it helps is by containing damage by limiting what an attacker can access or do with a compromised account. |
| 1. Sanitize Data Sent to Other Systems | Aims to sanitize outbound data is to ensure that the information sent to other systems or interfaces is safe, properly formatted, and free from potentially malicious content. The context in which this data is sent and received is important as well. For example, data sent to SQL queries, XML parsers, and LDAP queries each require different sanitization techniques. |
| 1. Practice Defense in Depth | In-depth defense means taking advantage of multiple layers of defense so that if one layer fails, another layer can prevent or reduce the effects of a successful attack. Some layers of defense include firewalls, encryption, access controls, intrusion detection systems, and secure coding practices. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance means thorough and comprehensive testing is done throughout the development process. This helps identify potential vulnerabilities and ensures that security controls are working correctly. This process involves creating security-focused test cases, threat modeling, and fuzz testing. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard means implementing a set of guidelines and best practices to ensure that software is developed with security in mind. This means following standards specifically designed for your coding language and release platforms. Doing so helps to minimize vulnerabilities and guards against potential cyber threats. |

### 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** | DCL31-C | Declare identifiers before using them. |

| **Noncompliant Code** |
| --- |
| This code omits the type specifier. |
| **extern** foo; |

| **Compliant Code** |
| --- |
| This code explicitly includes a type specifier. |
| **extern** **int** foo; |

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

| **Principles(s):** 2. Principle of Fail-safe defaults. Variables should be declared with a storage duration that ensures they are initialized predictably and have a well-defined lifetime. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang Static Analyzer | 15.0.0 | Alpha.core.CallAndMessage, alpha.core.DynamicTypeChecker | A source code analysis tool that finds bugs in C, C++, and Objective-C programs. |
| Cppcheck | [Insert text.] | uninitvar | A static analysis tool for C and C++ code that focuses on detecting bugs and potential issues like uninitialized variables and improper storage duration. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | NUM04-J | Do not use floating-point numbers if precise computation is required. |

| **Noncompliant Code** |
| --- |
| Performs basic currency calculations. |
| **double** dollar = 1.00;  **double** dime = 0.10;  **int** number = 7;  System.out.println(    "A dollar less " + number + " dimes is $" + (dollar - number \* dime)  ); |

| **Compliant Code** |
| --- |
| Uses integer type and works with cents rather than dollars. |
| **int** dollar = 100;  **int** dime = 10;  **int** number = 7;  System.out.println(    "A dollar less " + number + " dimes is $0." + (dollar - number \* dime)  ); |

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

| **Principles(s):** 1. Principle of Least Privilege. Used to reduce assumptions about the correctness or safety of operations, particularly those that can lead to unexpected or insecure results. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQUbe | 9.9 LTS | Java:S2111 | A widely used platform for continuous inspection of code quality that includes built-in checkers that can detect the improper use of floating-point arithmetic in Java code. |
| PMD | 6.55.0 | AvoidUsingFloat | An open-source static analysis tool that scans Java code for potential issues, including the misuse of floating-point numbers. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR05-C | Use pointers to const when referring to string literals. |

| **Noncompliant Code** |
| --- |
| Const has been omitted resulting in the assignment being undefined. |
| **char** \*c = "Hello"; |

| **Compliant Code** |
| --- |
| Const-qualified which means that any attempt to assign them to a different value results in an error. |
| **const** **char** \*c = "Hello"; |

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

| **Principles(s):** 4. Principle of Complete Mediation. Ensures that every access to a resource is checked to prevent unauthorized or unintended actions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2023.6 | STRING\_OVERFLOW | A commercial static analysis tool that detects a wide range of software defects, including buffer overflows, memory corruption, and out-of-bounds string writes. |
| Fortify Static Code Analyzer | SCA 23.1 | Buffer Overflow, Out-of-bounds-Write | A static code analysis tool that includes checkers that detect potential buffer overflows and out of bound writes. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | OWASP-ASVS | Application security verification standard |

| **Noncompliant Code** |
| --- |
| Concatenates user input directly into the SQL query, making it vulnerable to SQL injection attacks. |
| **public** User getUserByUsername(String username) {  String query = "SELECT \* FROM users WHERE username = '" + username + "'";  ResultSet rs = statement.executeQuery(query);  **return** rs.next() ? **new** User(rs.getInt("id"), rs.getString("username")) : **null**;  } |

| **Compliant Code** |
| --- |
| Uses a prepared statement to prevent SQL injection |
| **public** User getUserByUsername(String username) {  String query = "SELECT \* FROM users WHERE username = ?";  **try** (PreparedStatement pstmt = connection.prepareStatement(query)) {  pstmt.setString(1, username);  ResultSet rs = pstmt.executeQuery();  **return** rs.next() ? **new** User(rs.getInt("id"),  rs.getString("username")) : **null**;  }  } |

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

| **Principles(s):** 2. Principle of Fail-Safe Defaults. Ensures that the system remains secure by default and only allows specific, controlled actions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| OWASP ZAP | 2.15.0 | SQL Injection Scanner | An open-source security scanner that includes an active scanner that detects SQL injection vulnerabilities by sending various payloads and analyzing responses to identify possible injection points. |
| Burp Suite | Professional 2024.3 | SQL Injection Scanner | The Professional version includes an advanced scanner that detects SQL injection vulnerabilities by crawling web applications and testing for common and complex SQL injection techniques. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM01-C | Store a new value in pointers immediately after free() |

| **Noncompliant Code** |
| --- |
| The TYPE of a message is used to determine how to process the message itself. |
| **char** \*message;  **int** message\_type;    /\* Initialize message and message\_type \*/    **if** (message\_type == value\_1) {    /\* Process message type 1 \*/  **free**(message);  }  /\* ...\*/  **if** (message\_type == value\_2) {     /\* Process message type 2 \*/  **free**(message);  } |

| **Compliant Code** |
| --- |
| Setting message to NULL after it’s freed eliminates the possibility that the message pointer can be used to free the same memory more than once. |
| **char** \*message;  **int** message\_type;    /\* Initialize message and message\_type \*/    **if** (message\_type == value\_1) {    /\* Process message type 1 \*/  **free**(message);    message = NULL;  }  /\* ... \*/  **if** (message\_type == value\_2) {    /\* Process message type 2 \*/  **free**(message);    message = NULL;  } |

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

| **Principles(s):** 2. Principle of Fail-Safe Defaults. Ensures that the system defaults to a safe state and only allows explicit actions that are safe and controlled. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2023.6 | UNINITIALIZED\_MEMORY | A static analysis tool that can detect the use of functions prone to buffer overflows and suggest safer alternatives. |
| Clang Static Analyzer | 15.0.0 | alpha.core.Uninitialized | Includes checkers for detecting buffer overruns and unsafe memory operations |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | EXP06-J | Expressions used in assertions must not produce side effects |

| **Noncompliant Code** |
| --- |
| Attempts to delete all of the null names from the list in an assertion. |
| **private** ArrayList<String> names;    **void** process(**int** index) {  **assert** names.remove(**null**); // Side effect    // ...  } |

| **Compliant Code** |
| --- |
| Decouples the Boolean expression from the assertion. |
| **private** ArrayList<String> names;    **void** process(**int** index) {  **boolean** nullsRemoved = names.remove(**null**);  **assert** nullsRemoved; // No side effect    // ...  } |

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

| **Principles(s):** 7. Principle of Least Astonishment. Recommends that assertions should not be used for critical argument validation or runtime checks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Checkstyle | 10.3 | IllegalCatch | A static code analysis tool that can be configured to identify patterns where assertions are misused for argument validation. |
| SonarQube | 10.1 | Java:S2699 | A static analysis tool that can detect cases where assertions are used for argument validation. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR-56-CPP | Guarantee exception safety |

| **Noncompliant Code** |
| --- |
| Shows a flawed copy assignment operator. |
| #include <cstring>    **class** IntArray {  **int** \*array;    std::**size\_t** nElems;  **public**:    // ...      ~IntArray() {  **delete**[] array;    }        IntArray(**const** IntArray& that); // nontrivial copy constructor    IntArray& operator=(**const** IntArray &rhs) {  **if** (**this** != &rhs) {  **delete**[] array;        array = nullptr;        nElems = rhs.nElems;  **if** (nElems) {          array = **new** **int**[nElems];          std::**memcpy**(array, rhs.array, nElems \* **sizeof**(\*array));        }      }  **return** \***this**;    }      // ...  }; |

| **Compliant Code** |
| --- |
| The copy assignment operator offers the strong exception safety guarantee. |
| #include <cstring>    **class** IntArray {  **int** \*array;    std::**size\_t** nElems;  **public**:    // ...      ~IntArray() {  **delete**[] array;    }      IntArray(**const** IntArray& that); // nontrivial copy constructor      IntArray& operator=(**const** IntArray &rhs) {  **int** \*tmp = nullptr;  **if** (rhs.nElems) {        tmp = **new** **int**[rhs.nElems];        std::**memcpy**(tmp, rhs.array, rhs.nElems \* **sizeof**(\*array));      }  **delete**[] array;      array = tmp;      nElems = rhs.nElems;  **return** \***this**;    }      // ...  }; |

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

| **Principles(s):** 7. Principle of Least Astonishment. Recommends that exceptions should be caught by const reference to avoid potential modifications and ensure predictability in exception handling. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 15.0.0 | Readability-const-return-type | The readability-const-return-type checker helps identify instances where non-const references are used for exception handling, which can be adapted to ensure exceptions are caught by const reference. |
| Cppcheck | 2.10 | Style | The style checker helps ensure that code follows best practices, including catching exceptions by const reference to prevent unintended modifications. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Thread Safety | TSM00-J | Do not override thread-safe methods with methods that are not thread-safe. |

| **Noncompliant Code** |
| --- |
| Overrides the synchronized doSomething() method with an unsynchronized method. |
| **class** Base {  **public** **synchronized** **void** doSomething() {      // ...    }  }    **class** Derived **extends** Base {    @Override **public** **void** doSomething() {      // ...    }  } |

| **Compliant Code** |
| --- |
| Synchronizes the doSomething() method of the subclass. |
| class Base {  public synchronized void doSomething() {  // ...  }  }    class Derived extends Base {  @Override public synchronized void doSomething() {  // ...  }  } |

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

| **Principles(s):** 7. Principle of Least Astonishment. Recommends thread-safe access to shared mutable data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| FindBugs/SpotBugs | 4.5.0 | THREAD\_SAFETY | The THREAD\_SAFETY checker helps detect common thread safety problems, such as unsynchronized access to shared mutable data, and provides insights for improving thread safety. |
| Find Security Bugs | 1.11.0 | THREAD\_SAFETY | The THREAD\_SAFETY checker in this tool detects thread safety issues, including improper synchronization and unsafe access to shared data. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Serialization | SER03-J | Do not serialize unencrypted sensitive data |

| **Noncompliant Code** |
| --- |
| The data members within the Point class are private. Assuming the coordinates are sensitive, their presence in the stream would expose them to malicious tampering. |
| public class Point implements Serializable {  private double x;  private double y;    public Point(double x, double y) {  this.x = x;  this.y = y;  }    public Point() {  // No-argument constructor  }  }    public class Coordinates extends Point {  public static void main(String[] args) {  FileOutputStream fout = null;  try {  Point p = new Point(5, 2);  fout = new FileOutputStream("point.ser");  ObjectOutputStream oout = new ObjectOutputStream(fout);  oout.writeObject(p);  } catch (Throwable t) {  // Forward to handler  } finally {  if (fout != null) {  try {  fout.close();  } catch (IOException x) {  // Handle error  }  }  }  }  } |

| **Compliant Code** |
| --- |
| This code avoids the possibility of incorrect serialization and protects sensitive data by declaring the relevant members as transient. |
| **public** **class** Point **implements** Serializable {  **private** **transient** **double** x; // Declared transient  **private** **transient** **double** y; // Declared transient    **public** Point(**double** x, **double** y) {  **this**.x = x;  **this**.y = y;   }    **public** Point() {     // No-argument constructor   }  }    **public** **class** Coordinates **extends** Point {  **public** **static** **void** main(String[] args) {      FileOutputStream fout = **null**;  **try** {        Point p = **new** Point(5,2);        fout = **new** FileOutputStream("point.ser");        ObjectOutputStream oout = **new** ObjectOutputStream(fout);        oout.writeObject(p);        oout.close();      } **catch** (Exception e) {        // Forward to handler      } **finally** {  **if** (fout != **null**) {  **try** {            fout.close();          } **catch** (IOException x) {            // Handle error          }        }      }    }  } |

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

| **Principles(s):** 8. Principle of Defense Depth. Implement custom serialization and use additional security measures like encryption and serialization. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| FindBugs/SpotBugs | 4.5.0 | SERIALIZATION | The SERIALIZATION checker helps identify classes that use default serialization mechanisms and may require custom serialization implementations to address potential security risks. |
| SonarQube | 10.1 | Java:S2078 | The java:S2078 rule helps detect improper use of serialization mechanisms, including default serialization, and ensures that classes implementing Serializable use secure serialization practices. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Platform Security | SEC02-J | Do not base security checks on untrusted sources |

| **Noncompliant Code** |
| --- |
| A security vulnerability from the Java .io package allows an attacker to supply an untrusted argument by extending the legitimate File class. |
| public RandomAccessFile openFile(final java.io.File f) {  askUserPermission(f.getPath());  // ...  return (RandomAccessFile)AccessController.doPrivileged(new PrivilegedAction <Object>() {  public Object run() {  return new RandomAccessFile(f, f.getPath());  }  });  } |

| **Compliant Code** |
| --- |
| This code ensures that the java.io.File object can be trusted even though it’s not final. |
| public RandomAccessFile openFile(java.io.File f) {  final java.io.File copy = new java.io.File(f.getPath());  askUserPermission(copy.getPath());  // ...  return (RandomAccessFile)AccessController.doPrivileged(new PrivilegedAction <Object>() {  public Object run() {  return new RandomAccessFile(copy, copy.getPath());  }  });  } |

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

| **Principles(s):** 1. Principle of Least Privilege. Recommends the use of secure platform-specific APIs and features. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Checkmarx | 8.0 | Custom Rules | Checkmarx is a comprehensive static application security testing tool that helps identify vulnerabilities associated with insecure or deprecated APIs and ensures that secure platform-specific features are used. |
| FindBugs/Spotugs | 4.5.0 | SECURITY | The SECURITY category contains checks for insecure API usage, helping developers identify and replace outdated or insecure platform-specific APIs with more secure alternatives. |

### Defense-in-Depth Illustration

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



### 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.

Integrating Automation tools will play a crucial role in the transition process from DevOPs to a DevSecOps framework. For instance, integrating SonarQube and SonarLint into the development and continuous integration (CI) environments will ensure real-time feedback on code quality and adherence to security standards, automating the detection of vulnerabilities such as SQL Injection or buffer overflows. Tools like **OWASP ZAP** for dynamic application security testing (DAST) and **Contrast Security** for interactive application security testing (IAST) can be automated within the CI/CD pipeline to continuously scan for security issues during the build and deployment phases. Additionally, **Chaos Monkey** and **fuzzing** tools will automate resilience testing by simulating failures and unexpected inputs.

For runtime security, integrating **Runtime Application Self-Protection (RASP)** tools and implementing automated **security orchestration** with platforms like **SOAR** will provide real-time threat detection and response. Automated **software signing** and **integrity checks** will ensure that code remains secure throughout its lifecycle. By incorporating these tools into the DevSecOps pipeline, Green Pace will enhance its ability to enforce security standards, automate compliance, and maintain a robust security posture while minimizing manual intervention and operational overhead.

### 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 |
| --- | --- | --- | --- | --- | --- |
| EXP06-J | Medium | Medium | Low to Medium | Medium | 2 |
| DCL31-C | High | Medium | Medium | High | 1 |
| NUM04-J | Medium to High | Medium | Low to Medium | High | 2 |
| MEM01-C | High | Medium to High | Medium | High | 1 |
| SER03-J | High | Medium | Medium | High | 1 |
| ERR-56-CPP | Medium | Medium | Low to Medium | Medium | 2 |
| OWASP-ASVS | High | High | Medium to High | High | 1 |
| STR05-C | High | Medium to High | Low to Medium | High | 1 |
| TSM00-J | High | Medium to High | Medium to High | High | 1 |
| SEC02-J | High | Medium | Medium | High | 1 |

### 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 | Encrypts data stored on disks and backups by implementing AES encryption and secure key management. This is done to protect data from unauthorized access if the media becomes compromised. |
| Encryption in flight | Encrypts data during the transmission across networks by using protocols like TLS and S/MIME. This prevents unauthorized interceptions and eavesdropping. |
| Encryption in use | Encrypts data while it is being processed by applying homomorphic encryption or secure enclaves. This secures data during computation and prevents exposure. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Verifies user or system identities by using multi-factor authentication and IAM systems. Ensures only authorized entities access resources. |
| Authorization | Controls permissions for accessing resources by implementing role-based or attribute-based access controls. Enforces least privilege and limits access to authorized users. |
| Accounting | Tracks and records user actions and system activities by logging and monitoring solutions with audit reporting. Provides visibility for detecting anomalies and ensuring compliance. |

**\***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 | 7/20/2024 | Coding Standards | Chance Roy | [Insert text.] |
| 3.0 | 8/9/2024 | Coding Principles | Chance Roy | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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