**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 from every source which can help prevent any input vulnerabilities from being exploited. |
| 1. Heed Compiler Warnings | Carefully analyze and fix all warnings generated when compiling the code. This can help prevent any security flaws from being exploited. |
| 1. Architect and Design for Security Policies | Create and implement the system architecture in order to enforce specific security policies. This can help strengthen the system's security. |
| 1. Keep It Simple | Keep the design simple to avoid any errors that can arise from having a large and complex system. This can help prevent any errors but also simplify the complexity of security needed. |
| 1. Default Deny | By default ensure that permissions are denied to users. This can help ensure users only have access to actions they are permitted to do. |
| 1. Adhere to the Principle of Least Privilege | Ensure that the least amount of access privileges are given to complete the task. This prevents helps prevent attackers from gaining higher privileges through their attacks. |
| 1. Sanitize Data Sent to Other Systems | Ensure that all data used in databases is sanitized beforehand. This helps prevent SQL injections. |
| 1. Practice Defense in Depth | Manage security by implementing multiple layers of defense strategies. This helps ensure that even if one layer fails, the other continues securing the application. |
| 1. Use Effective Quality Assurance Techniques | Ensure proper testing techniques are used such as penetration testing. This helps identify any security concerns that may have been overlooked. |
| 1. Adopt a Secure Coding Standard | Create or use an existing standard for the development language being used. |

### 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-CPP] | Ensure that the correct data type is used for all variables. |

| **Noncompliant Code** |
| --- |
| Ensure that the correct data type is used for all variables. |
| int num = “123”; |

| **Compliant Code** |
| --- |
| Assigning an integer to a integer type variable |
| int num = 123; |

| **Principles(s):** Principle 1: Validate Input Data  Principle 2: Least Privilege |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.16.0 | Type mismatch | Static analysis tool that ensures variables are using correct data types |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Validate all inputs before they are used |

| **Noncompliant Code** |
| --- |
| UserInput is not validated before used |
| int age = userInput; |

| **Compliant Code** |
| --- |
| This is required before age variable is used in order to validate age input is correct |
| if (age >= 0 && age <= 120) {} |

| **Principles(s):** Principle 1: Validate Input Data  Principle 6: Availability |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 20.0 | cppcoreguidlines-pro-bounds-pointer-arithmetic | Flags invalidated user inputs that are being used |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Ensure strings are handled correctly to prevent buffer overflow |

| **Noncompliant Code** |
| --- |
| String causing buffer overflow |
| char str[10];  strcpy(str, “This string is too long and will cause buffer overflow”); |

| **Compliant Code** |
| --- |
| Using strncpy to ensure there is no buffer overflow |
| char str[50];  strncpy(str, “This string is okay”, sizeof(str) – 1); |

| **Principles(s):** Principle 2: Heed Compiler Warnings  Principle 4: Keep It Simple  Principle 8: Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.16.0 | Buffer Overflow | Detects buffer overflows |
| Clang-Tidy | 20.0 | Clang-analyzer-security.insecure-strcpy | Flags unsafe string operations |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Defend against SQL Injection py making sure queries are correct |

| **Noncompliant Code** |
| --- |
| Assigning user input directly to a query allows SQL injection attacks |
| string query = “SELECT \* FROM users WHERE username = ‘” + userInput + “’;”; |

| **Compliant Code** |
| --- |
| Uses prepares statement to prevent SQL injection attacks |
| string query = “SELECT \* FROM users WHERE username = ?”;  PreparedStatement stmt = conn.prepareStatement(query);  stmt.setString(1, userInput); |

| **Principles(s):** Principle 1: Validate Input Data  Principle 3: Architect and Design for Security Policies  Principle 8: Defense In Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Checkmarx | 9.6.0 | SQL Injection | Checks applications for SQL injection vulnerabilities. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Ensure memory is accessed correctly. |

| **Noncompliant Code** |
| --- |
| Writing past allowed memory can cause errors |
| char\* ptr = new char[100];  prt[150] = ‘A’; |

| **Compliant Code** |
| --- |
| Ensure the memory bounds are checked before assignment |
| char\* ptr = new char[100];  if (index < 100) {  ptr[index] = ‘A’;  } |

| **Principles(s):** Principle 1: Validate Input Data  Principle 5: Least Privilege  Principle 9: Monitoring and Logging |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 20.0 | cppcoreguidelines-pro-bounds-pointer-arithmetic | Detects out-of-bounds memory access |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use assertions to catch errors |

| **Noncompliant Code** |
| --- |
| Not checking if dividing by 0 |
| int result = divide(x, y); |

| **Compliant Code** |
| --- |
| Asserting that division by 0 does not occur |
| assert(y !=0);  int result = divide(x, y); |

| **Principles(s):** Principle 2: Heed Compiler Warnings  Principle 7: Sanitize Data Sent to Other Systems |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 20.0 | bugprone-assert-side-effect | Ensure assertions do not cause side effects. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Properly handle all exceptions |

| **Noncompliant Code** |
| --- |
| Not handling a error caused when dividing by 0 |
| int divide(int x, int y) {  return x / y;  } |

| **Compliant Code** |
| --- |
| Throws exception when dividing by 0 |
| int divide(int x, int y) {  if (y == 0) {  throw std::invalid\_argument(“Division by zero”);  }  return x / y;  } |

| **Principles(s):** Principle 6: Availability  Principle 8: Defense in Depth |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 2.16.0 | Uncaught Exception | Flags exceptions that are thrown but not caught. |

### 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.

One place the existing DevOps process could be modified to automate the enforcement of the standards in this security policy is by including unit tests for the exception and assertion coding standards within the Verify and test portion of the above DevSecOps diagram. Furthermore, another place the existing DevOps process could be modified to automate the enforcement of the standards in this security policy is by implementing static analysis tools such as Cppcheck and Clang-Tidy during the build process shown in the above DevSpeOps diagram. Therefore, this would help enforce coding standards such as data types and data values. Finally, one last place the existing DevOps process could be modified to automate the enforcement of the coding standard in this security policy is by properly testing for memory protection throughout the monitor and detect phase shown in the above DevSecOps diagram, which could allow memory issues to be detected and found by monitoring the application with analysis tools.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Medium | Low | High | 2 |
| STD-002-CPP | High | Medium | Low | High | 2 |
| STD-003-CPP | High | High | Low | High | 2 |
| STD-004-CPP | High | High | Medium | High | 2 |
| STD-005-CPP | High | Medium | Low | High | 2 |
| STD-006-CPP | High | Medium | Low | Medium | 1 |
| STD-007-CPP | High | Medium | Medium | Medium | 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 | Encryption at rest is data that is encrypted while it is stored somewhere such as a database. This policy is best applied when storing sensitive data because it ensures that data is protected even if the database is breached. |
| Encryption in flight | Encryption at rest is data that is encrypted while it is being transferred over the network. This policy is best applied when transferring sensitive data such as login information because it ensures that the data is encrypted as it moves between the user and server. |
| Encryption in use | Encryption in use is data that is encrypted while it is being used and while it is stored in memory. This policy is best applied when accessing sensitive data such as when preforming database interactions because it ensures that all data is encrypted while being manipulated. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication ensures whoever is using a system is supposed to be using it. |
| Authorization | Authorization determines what actions a user is allowed to make. |
| Accounting | Accounting tracks user activities. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [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 |