**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 | Verify data received from users, files, and network connections. Not doing so could cause crashes or injection attacks. |
| 1. Heed Compiler Warnings | All warnings should be considered errors. If ignored, the program could then have bugs or practices that are not ideal. |
| 1. Architect and Design for Security Policies | Architecture design plays a big role in secure development. We can only secure what has a good foundation and the architecture of the project is key. Identifying threats should occur throughout the entire project process, not just added in at the end. |
| 1. Keep It Simple | Security is best when a project can be revisited at a later time and be completely understood. Consistency is important in keeping projects secure. The project should be easily maintained over time. |
| 1. Default Deny | This pertains to the strictness of the program. The project should automatically restrict access to anyone except users that have been given access on purpose. This process defaults security instead of allowing security measures to be an afterthought. |
| 1. Adhere to the Principle of Least Privilege | When incorporated with default deny, this process specifically only allows users to perform necessary actions. Users aren’t allowed do things that aren’t a specific priority. |
| 1. Sanitize Data Sent to Other Systems | All data should be reviewed on a regular basis. Monitoring and looking for injections or items that should not be included is important. Sanitizing data makes sure all data is secure and regularly monitored. |
| 1. Practice Defense in Depth | This policy requires that projects have multiple layers of security in play. Overlapping layers of security is the most thorough way of making sure a project is safe from grey or black hat hackers. |
| 1. Use Effective Quality Assurance Techniques | Utilizing specific techniques are important in acting out a defense in depth policy. The multiple layers of security should be regularly monitored and run through quality assurance techniques like unit testing or code reviews. |
| 1. Adopt a Secure Coding Standard | Following a specific standard allows for consistency. Consistency can be achieved if the same policies or rules are being used. Consistency allows for better quality control and ongoing maintenance which heightens security measures over time. |

### 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 | The Data Type Coding Standard is when you use the correct type of data to represent the data. Using the wrong data type can cause errors and the potential for vulnerabilities in security measures.  INT02-CPP |

| **Noncompliant Code** |
| --- |
| Using a number in the char |
| Char = 34; |

| **Compliant Code** |
| --- |
| Using the correct data type allows for the use of defense in depth and in turn secure code. |
| Int = 34; |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior |
| Coverity | Latest | INT Conversion | Detects conversion |
| CPP Check | 2.12 | Information | Warns about conversions |
| SonarQube | Latest | S1481 | Flags Incorrect Values |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | The Data Value Coding Standard monitors integer values within the code. INT 31-CPP |

| **Noncompliant Code** |
| --- |
| Integer values not being monitored allows for potential truncation |
| Int data = input; |

| **Compliant Code** |
| --- |
| Integer values fitting in boundaries allows for defense in depth. |
| Int data = any given value <int>(input); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior |
| Coverity | Latest | INT Conversion | Detects conversion |
| CPP Check | 2.12 | Information | Warns about conversions |
| SonarQube | Latest | S1481 | Flags Incorrect Values |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | String Correctness Coding Standard prevents strings being created from null pointers. STR50-CPP |

| **Noncompliant Code** |
| --- |
| No monitoring of strings |
| std::string s(str); |

| **Compliant Code** |
| --- |
| Making sure strings aren’t created out of null pointers |
| If (str = nullptr) … |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Low | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior |
| Coverity | Latest | INT Conversion | Detects conversion |
| CPP Check | 2.12 | Information | Warns about conversions |
| SonarQube | Latest | S1481 | Flags Incorrect Values |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Prevent SQL injection through sanitation and input validation FI030-C |

| **Noncompliant Code** |
| --- |
| Allow SQL Injection |
| example permits a SQL injection attack by incorporating the unsanitized input argument username into the SQL command, allowing an attacker to inject validuser' OR '1'='1. |

| **Compliant Code** |
| --- |
| parametric query with a ? character as a placeholder for the argument. |
| String sqlString =        "select \* from db\_user where username=? and password=?";      PreparedStatement stmt = connection.prepareStatement(sqlString);      stmt.setString(1, username);      stmt.setString(2, pwd); |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | Latest | S2077 | Detects SQL queries |
| Fortify | Latest |  | Detects SQL Injection Vulnerabilites |
| Code QL | Latest |  | Analyzes SQL Logic |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Avoid Memory Leaks MEM 50-CPP. Properly manage allocated memory. |

| **Noncompliant Code** |
| --- |
| Memory never freed |
| Int\* ptr = new int(5); |
|  |

| **Compliant Code** |
| --- |
| Ensure memory managed automatically |
| #include <memory>  Std::unique\_ptr<int> ptr = std::make\_unique<int>(5); |

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

| **Principles(s):** Quality Assurance is to be followed |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ClangStatic | 16 | unix | Detects unallocated memory |
| Valgrind | Latest | MemCHECK | Runtime management |
| Cpp Check | 2.12 |  | Manages memory leaks |
|  |  |  |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Avoid using assertions for function arguments EXP05-CPP |

| **Noncompliant Code** |
| --- |
| (**use assertion for check function** |
| Void setIndex(int index) {  assert(index >= 0);  this ->index = index;  } |

| **Compliant Code** |
| --- |
| Uses runtime exception |
| Void setIndex(int index) {  If (index < 0) {  Throw std::invalid)argument(“Index must be non-negative”);  { |

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

| **Principles(s):** Validate input data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | High | Low | Low | 4 |

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior |

|  |  |  |  |
| --- | --- | --- | --- |
| SonarQube | Latest | S2077 | Detects SQL queries |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Guarantee exception safety ERR50-CPP |

| **Noncompliant Code** |
| --- |
| Exceptions not handled correctly |
| Try {  performAction();  } catch |

| **Compliant Code** |
| --- |
| Catches exceptions |
| Try {  performAction();  } catch (const std::exception& e) {  Std::cerr << “Error: “ << e.what() << std::endl;  } |

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

| **Principles(s):** Adopt secure coding standard |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | MEdium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| | **Tool** | **Version** | **Checker** | **Description Tool** | | --- | --- | --- | --- | | Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior | | | **Tool** | **Version** | **Checker** | **Description Tool** | | --- | --- | --- | --- | | Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior | | | **Tool** | **Version** | **Checker** | **Description Tool** | | --- | --- | --- | --- | | Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior | | | **Tool** | **Version** | **Checker** | **Description Tool** | | --- | --- | --- | --- | | Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior | |
| |  |  |  |  | | --- | --- | --- | --- | | SonarQube | Latest | S2077 | Detects SQL queries | | |  |  |  |  | | --- | --- | --- | --- | | SonarQube | Latest | S2077 | Detects SQL queries | | |  |  |  |  | | --- | --- | --- | --- | | SonarQube | Latest | S2077 | Detects SQL queries | | |  |  |  |  | | --- | --- | --- | --- | | SonarQube | Latest | S2077 | Detects SQL queries | |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | STD-008-CPP | Range Check STR53-CPP avoid undefined bahavior |

| **Noncompliant Code** |
| --- |
| Out of range index |
| Std::string str = “hello”;  Char ch = str[10]; |

| **Compliant Code** |
| --- |
| Checks the index |
| Std::string str = “hello”;  If (index < str.size()) {  Char ch = str[index];  } |

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

| **Principles(s):** Validate Input Data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| | **Tool** | **Version** | **Checker** | **Description Tool** | | --- | --- | --- | --- | | Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior | | | **Tool** | **Version** | **Checker** | **Description Tool** | | --- | --- | --- | --- | | Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior | | | **Tool** | **Version** | **Checker** | **Description Tool** | | --- | --- | --- | --- | | Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior | | | **Tool** | **Version** | **Checker** | **Description Tool** | | --- | --- | --- | --- | | Clang-Tidy | 16 | Clang Analyzer Core | Detects behavior | |
| |  |  |  |  | | --- | --- | --- | --- | | SonarQube | Latest | S2077 | Detects SQL queries | | |  |  |  |  | | --- | --- | --- | --- | | SonarQube | Latest | S2077 | Detects SQL queries | | |  |  |  |  | | --- | --- | --- | --- | | SonarQube | Latest | S2077 | Detects SQL queries | | |  |  |  |  | | --- | --- | --- | --- | | SonarQube | Latest | S2077 | Detects SQL queries | |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Release Resources | STD-009-CPP | Functions are to release resources ERR34-CPP |

| **Noncompliant Code** |
| --- |
| Resources not released |
| FILE\* f = fopen(“data.txt”, “r”); |

| **Compliant Code** |
| --- |
| Automatically releases resources |
| Std::ifstream file(“data.txt”); |

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

| **Principles(s):** Use quality assurance techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ClangStatic | 16 | unix | Detects missing resources |
| Code Sonar | Latest |  | Detects unreleased resources |

#### 

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | STD-010-CPP | Do not use deprecated functions MSC04-CPP |

| **Noncompliant Code** |
| --- |
| Deprecated function |
| Gets(buffer); |

| **Compliant Code** |
| --- |
| Safe use C++ use |
| Std::getline(std::cin, buffer); |

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Medium | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ClangStatic | 16 | unix | Detects missing resources |
| Code Sonar | Latest |  | Detects unreleased resources |

### Defense-in-Depth Illustration

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



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation is important to be performed at multiple times in the SDLC. These stages are to be spread out over time or the lifetime of the project. Using threat modeling and planning the stages in important at this step of the project.

### Summary of Risk Assessments

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

Summary of Risk Assessments

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Rule** | **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Medium | Low | High | 4 |
| STD-003-CPP | High | High | Low | Very High | 5 |
| STD-004-CPP | Critical | High | Low | Very High | 5 |
| STD-005-CPP | High | Medium | Medium | High | 4 |
| STD-006-CPP | Medium | High | Low | High | 4 |
| STD-007-CPP | Medium | Medium | Medium | Medium | 3 |
| STD-008-CPP | High | Medium | Low | High | 4 |
| STD-009-CPP | High | Medium | Medium | High | 4 |
| STD-010-CPP | Medium | Medium | Low | Medium | 3 |

### 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 | Encrypt all data on discs |
| Encryption in flight | TLS 1.2 should be used for confidentiality |
| Encryption in use | Runtime protections to be used for sensitive data |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Verify user identity |
| Authorization | Enforce role based access only |
| Accounting | Record all logins |

**\***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 |