**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 | Test input to ensure correct data enters the system, and malicious data is prevented. |
| 1. Heed Compiler Warnings | Used to notify the developer of errors when compiling code. Warnings are used to notify before potential errors occur, preventing unnecessary issues. |
| 1. Architect and Design for Security Policies | Architecture and design must be considered when implementing security policies, such as implementing role-specific authorization. |
| 1. Keep It Simple | Reduce complications in code to prevent confusion and unnecessary issues. |
| 1. Default Deny | Access is denied by default. |
| 1. Adhere to the Principle of Least Privilege | Processes should be executed using minimal required privileges in order to prevent attacks through the use of elevated access to the system. |
| 1. Sanitize Data Sent to Other Systems | Clean data before passing it onto other systems through various checks to prevent issues between systems. |
| 1. Practice Defense in Depth | Use multiple layers of defense to prevent exploits and malicious attacks. |
| 1. Use Effective Quality Assurance Techniques | Proper testing through internal and external security reviews can prevent potential issues. Proper QA programs can help to secure the systems throughout the development process. |
| 1. Adopt a Secure Coding Standard | Apply coding standards and best practices based on coding language and platform. |

### 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] | Obey one-definition rule |

| **Noncompliant Code** |
| --- |
| Different translation units define a class of the same name with differing definitions. |
|  |

| **Compliant Code** |
| --- |
| Use of a header file to introduce the object into both translation units. |
|  |

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

| **Principles(s):**   * Adopt a Secure Coding Standard * Architect and Design for Security Policies * Keep it Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ Test | 2021 | CERT\_CPP-DCL60-a | Unique identifiers must be used for a class, union, or enum name. |
| Axivion Bauhaus Suite | 7.2 | CertC++-DCL60 |  |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Implemented Fully |
| Astree | 20.10 | Type-file-spreading  External-file-spreading  Type-compatibility | Partially checked |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized memory. |

| **Noncompliant Code** |
| --- |
| Uninitialized local variable is evaluated as part of an expression to print value, resulting in undefined behavior. |
|  |

| **Compliant Code** |
| --- |
| Object is initialized prior to printing its value. |
|  |

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

| **Principles(s):**   * Adopt a Secure Coding Standard * Validate Input Data * Keep It Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2021a | CERT C++: EXP53-CPP | Non-initialized pointer  Non-initialized variable |
| Astree | 20.10 | Uninitialized-Read | Partially Checked |
| LDRA Tool Suite | 9.7 | 53 D, 69 D, 631S, 652S | Implemented Partially |
| Helix QAC | 2021 | C++2726, C++2978, C++2728 | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create std::string from a null pointer. |

| **Noncompliant Code** |
| --- |
| A std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior within the environment. |
|  |

| **Compliant Code** |
| --- |
| The results from the call to std::getenv() are checked for null before the std::string object is constructed. |
|  |

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

| **Principles(s):**   * Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft | 2021 | CERT\_CPP-STR51-a | Avoid Dereferencing Null Pointer |
| KlocWork | 2021 | NPD.CHECK.CALL.MIGHT  NPD.CHECK.MUST  NPD.CONST.CALL  NPD.FUNC.CALL.MIGHT  NPD.GEN.CALL.MIGHT  RNPD.CALL  RNPD.DEREF |  |
| Astree | 20.10 | Assert\_failure |  |
| Helix QAC | 2021 | C++4470, - C++4774 |  |

#### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| Without precautions, the untrusted data may maliciously alter the query. |
|  |

| **Compliant Code** |
| --- |
| The primary means of preventing SQL injection are sanitization & validation, which are typically implemented as parameterized queries and stored procedures. |
|  |

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

| **Principles(s):**   * Adopt a Secure Coding Standard * Sanitize Data Sent to Other Systems * Validate Input Data |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented Fully |
| Parasoft Jtest | 2021.1 | CERT.IDS00.TDSQL | Protect against SQL Injection |
| Fortify | 1 | HTTP\_Response\_Splitting  SQL\_Injection | Implemented |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and Security Errors |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| Memory is dereferenced after it has been deallocated. |
|  |

| **Compliant Code** |
| --- |
| The dynamically allocated memory is not deallocated until it is no longer required. |
|  |

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

| **Principles(s):**   * Use Effective Quality Assurance Techniques * Default Deny * Heed Compiler Warnings * Adhere to the Principle of Least Privilege |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Insure++ |  |  | Runtime Detection |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP0-MEM50 | Do not use freed resources |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDelete | Checked by clang-tidy |
| Coverity | V7.5.0 | Use\_After\_Free | Detect instances where memory is deallocated more than once |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CLG] | Use static assertion to test the value of a constant expression. |

| **Noncompliant Code** |
| --- |
| Uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly. |
|  |

| **Compliant Code** |
| --- |
| For constant expressions, a preprocessor conditional statement may be used. |
|  |

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

| **Principles(s):**   * Adopt a Secure Coding Standard * Heed Compiler Warnings |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Éclair | 1.2 | CC2.DCL03 | Implemented Fully |
| LDRA Tool Suite | 9.7.0 | 44S | Implemented Fully |
| Axivion Bauhuas Suite | 7.2.0 | CertC-DCL03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by Clang-tidy |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not abruptly terminate the program. |

| **Noncompliant Code** |
| --- |
| The call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate(). |
|  |

| **Compliant Code** |
| --- |
| f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
|  |

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

| **Principles(s):**   * Adopt a Secure Coding Standard * Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klockwork | 2021.1 | MISRA.CATCH.ALL  CERT.ERR.ABRUPT\_TERM |  |
| CodeSonra | 6.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of Abort/Exit |
| LDRA Tool Suite | 9.7.1 | 122S | Enforcement |
| Polyspace Bug Finder | R2021a | CERT C++ERR50-CPP | Checks for implicit call to terminate function |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CPP] | Write constructor member initializers in the canonical order. |

| **Noncompliant Code** |
| --- |
| The member initializer list for C::C() attempts to initialize someVal first and then to initialize dependsOnSomeVal to a value dependent on someVal. Because the declaration order of the member variables does not match the member initializer order, attempting to read the value of someVal results in an unspecified value being stored into dependsOnSomeVal. |
|  |

| **Compliant Code** |
| --- |
| Change the declaration order of the class member variables so that the dependency can be ordered properly in the constructor's member initializer list. |
|  |

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

| **Principles(s):**   * Keep it Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Initializer-list-order | Checked Fully |
| LDRA Tool Suite | 9.7.1 | 206 S | Implemented Fully |
| Axivion Bauhuas Suite | 7.2.0 | CertC++-OOP53 |  |
| Parasoft C/C++ Test | 2021.1 | CERT\_CPP-OOP53-a | List members in the order they are declared |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | Use valid iterator ranges. |

| **Noncompliant Code** |
| --- |
|  |
|  |

| **Compliant Code** |
| --- |
| The iterator values passed to std::for\_each() are passed in the proper order. |
|  |

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

| **Principles(s):**   * Adopt a Secure Coding Standard * Keep it Simple * Architect and Design for Security Policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.14 | V539, V789, V662 |  |
| Parasoft C/C++ Test | 2021.1 | CERT\_CPP-CTR53-a  CERT\_CPP-CTR53-b | Do not compare iterators from different containers |
| Astree | 20.10 | Overflow\_upon\_dereference |  |
| Helix QAC | 2021.2 | C++3802 |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | Do not access an object outside of its lifetime. |

| **Noncompliant Code** |
| --- |
| A pointer to an object is used to call a non-static member function of the object prior to the beginning of the pointer's lifetime, resulting in undefined behavior. |
|  |

| **Compliant Code** |
| --- |
| Storage is obtained for the pointer prior to calling S::mem\_fn(). |
|  |

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

| **Principles(s):**   * Adopt a Secure Coding Standard * Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Return-reference-local | Partially checked |
| Clang | 3.9.0 | Wdanging-initializer-list | Finds issues related to use of std::initializer\_list<> |
| Parasoft C/C++Test | 2021.1 | CERT\_CPP-EXP54-a  CERT\_CPP-EXP54-b  CERT\_CPP-EXP54-c | Do not use resources that have been freed |
| CodeSonar | 6.1p0 | IO.UAC  ALLOC.UAF | Use after close/free |

### 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.

### 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 | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Probable | Medium | High | 1 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Probable | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CLG | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | Low | 3 |
| STD-009-CPP | High | Probable | High | High | 1 |
| STD-010-CPP | High | Probable | High | Medium | 2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Protects stored data through devices such as hard drives, sim cards, and cloud assets. |
| Encryption at flight | Protects data that is moving between devices inside or outside of a network. Common methods are email encryption, firewalls, and DLP solutions. |
| Encryption in use | Protects data that is created, edited, or defined as in-use. Data control/protection are typically done before data is created, this is typically seen in programs such as Microsoft Office |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Confirming one’s identity through certifications, biometrics, or passwords. Used to ensure the person trying to gain access is actually the authorized user. |
| Authorization | Authorization refers to the access rights and privileges of the user. Access rights and privileges are important security measures to ensure the user is only able to access data they require and cannot access restricted or sensitive information. |
| Accounting | Accounting manages user activity while interacting with the system. Accounting typically tracks timestamps, resources accessed, and data transferred. Used to follow user activity and habits. |

**\***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 |  |
| 1.1 | 09/24/2023 | Module 3 Milestone | Gregory Greene |  |
| 1.2 | 10/8/2023 | Module 6 Milestone | Gregory Greene |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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