**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 | All expected user input must be assumed malicious and never trusted. Always restrict user input to expected data type and check for data length especially when receiving string inputs. |
| 1. Heed Compiler Warnings | Never ignore compiler warnings of any level. All compiler warnings must be addressed by targeting and modifying affected code segments. |
| 1. Architect and Design for Security Policies | Always follow secure software architecture practices such as those outlined in the OWASP secure coding practices. Also load only secure external libraries and plugins and consistently look out for latest threat relating to them. |
| 1. Keep It Simple | Use simple and intuitive coding designs that are easy to debug by others. |
| 1. Default Deny | Avoid system use by default all users must be properly authenticated or permitted to use the system. |
| 1. Adhere to the Principle of Least Privilege | Users must only have privileged access to the data or system modules needed to successfully complete the tasks expected for their level of authorization and no more. |
| 1. Sanitize Data Sent to Other Systems | Use characters blacklist and whitelist to sanitize user input. Escape user input when necessary to avoid input from being interpreted as code. |
| 1. Practice Defense in Depth | Use overlapping and multiple redundant security layers to defend again multiple security threats. |
| 1. Use Effective Quality Assurance Techniques | Use unit tests and assertions to ensure code modules meets expected quality before integrating into system. |
| 1. Adopt a Secure Coding Standard | Adhere strictly to all the organization’s internal security policies and other globally accepted security policies. |

### 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] | Never qualify a reference type with const or volatile. |

| **Noncompliant Code** |
| --- |
| A const-qualified reference to a char is formed instead of a reference to a const-qualified char. |
| #include <iostream>    **void** f(**char** c) {  **char** &**const** p = c;    p = 'p';    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Solution removes the const qualifier. |
| #include <iostream>    **void** f(**char** c) {  **char** &p = c;    p = 'p';    std::cout << c << std::endl;  } |

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

| **Principles(s):** (1) Heed Compiler Warnings: Referencing a constant result in compiler diagnostic warnings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL52 |  |
| Helix QAC | 2023.3 | C++0014 |  |
| Klocwork | 2023.3 | **CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE** |  |
| Parasoft C/C++test | 2023.1 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| Attempt to check whether a given value is within the range of acceptable enumeration values after casting to the enumeration type. |
| **enum** EnumType {    First,    Second,    Third  };    **void** f(**int** intVar) {    EnumType enumVar = **static\_cast**<EnumType>(intVar);    **if** (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| Checks that the value can be represented by the enumeration type before performing the conversion using bound checking. |
| **enum** EnumType : **int** {    First,    Second,    Third  };    **void** f(**int** intVar) {    EnumType enumVar = **static\_cast**<EnumType>(intVar);  } |

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

| **Principles(s):** (1) ValidateInput Data to ensure proper scope for value type representation. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Cast-integer-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 8.0p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion Alters Value  Cast Alters Value |
| Helix QAC | 2023.3 | C++3013 | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| While it may be useful for a single string input, using the std::ios\_base::width() method can still cause overflow of the second string input. |
| #include <iostream>    **void** f() {  **char** bufOne[12];  **char** bufTwo[12];    std::cin.width(12);    std::cin >> bufOne;    std::cin >> bufTwo;  } |

| **Compliant Code** |
| --- |
| Guarding against buffer overflows using std::string instead of a bounded array |
| #include <iostream>  #include <string>    **void** f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):** (1) Validate input using string function (2) Sanitize string input to prevent again string attack. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Stream-input-char-array | Partially checked + soundly supported |
| CodeSonar | 8.0p0 | MISC.MEM.NTERM  LANG.MEM.BO LANG.MEM.TO | No space for null terminator  Buffer overrun Type overrun |
| Helix QAC | 2023.3 | C++5216  DF2835, DF2836, DF283 |  |
| Klocwork | 2023.3 | NNTS.MIGHT NNTS.TAINTED NNTS.MUST SV.UNBOUND\_STRING\_INPUT.CIN |  |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Never accept unvalidated or un-sanitized query strings. |

| **Noncompliant Code** |
| --- |
| Function retrieves records from an SQL database without query string sanitization or validation. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  records.clear();  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  std::cout << std::endl << "SQL: " << sql << " ==> " << records.size() << " records found." << std::endl;  for (auto record : records)  {  std::cout << "User: " << std::get<1>(record) << " [UID=" << std::get<0>(record) << " PWD=" << std::get<2>(record) << "]" << std::endl;  }  return true;  } |

| **Compliant Code** |
| --- |
| Function using a blacklist of bad character to check for SQL injection in query string. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  static char bad\_chars[] = "= /;[]<>&\t";    size\_t match = strcspn(sql.c\_str(), bad\_chars);  if (match < strlen(sql.c\_str())) {    std::cout << std::endl << "SQL: " << sql << std::endl;  std::cout << "QUERY STRING CONTAINS INJECTION!!!!" << std::endl;  return false;  }  records.clear();  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);    return false;  }  std::cout << std::endl << "SQL: " << sql << " ==> " << records.size() << " records found." << std::endl;  for (auto record : records)  {  std::cout << "User: " << std::get<1>(record) << " [UID=" << std::get<0>(record) << " PWD=" << std::get<2>(record) << "]" << std::endl;  }  return true;  } |

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

| **Principles(s):** (1) Sanitize input data avoid potential SQL injections. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus suite | 7.2.0 | CertC-STR31 | Detects calls to unsafe string function that may cause buffer overflow Detects potential buffer overruns, including those caused by unsafe usage of fscanf() |
| CodeSonar | 8.0p0 | LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\* | Buffer overrun Type overrun No space for null terminator A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| Helix QAC | 2023.4 | C2840,  C5009, C5038  C++0145, C++5009, C++5038  DF2840, DF2841, DF2842, DF2843, DF2845, DF2846, DF2847, DF2848, DF2930, DF2931, DF2932, DF2933, DF2935, DF2936, DF2937, DF2938 |  |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| Accessing memory location for s after it has been deallocated resulting in exploitable memory vulnerabilities and difficult problem diagnosis relating to memory allocation and deallocation. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| The dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** s;  } |

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

| **Principles(s):** (1) Architect and design for security by using proper memory allocation schemes. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Dangling\_pointer\_use | [Insert text.] |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM50 | [Insert text.] |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 8.0p0 | ALLOC.UAF | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Never use the assert() macro to verify that a memory allocation succeeded |

| **Noncompliant Code** |
| --- |
| Using the assert() macro to verify that a memory allocation succeeded would be inappropriate because doing so might lead to an abrupt termination of the process, opening the possibility of a denial-of-service attack. |
| **char** \*dupstring(**const** **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char** \*)**malloc**(len + 1);  **assert**(NULL != dup);    **memcpy**(dup, c\_str, len + 1);  **return** dup;  } |

| **Compliant Code** |
| --- |
| How to detect and handle possible memory exhaustion with the assert() macro |
| **char** \*dupstring(**const** **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char**\*)**malloc**(len + 1);    /\* Detect and handle memory allocation error \*/  **if** (NULL == dup) {  **return** NULL;    }    **memcpy**(dup, c\_str, len + 1);  **return** dup;  } |

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

| **Principles(s):** (1) Use effective quality assurance techniques to ensure proper assertions are used in the right context. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Coverity | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| Parasoft C/C++test | 2023.1 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| Neither f() nor main() catch exceptions thrown by throwing\_func(). |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| The main entry point handles all exceptions. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {  **try** {      f();    } **catch** (...) {      // Handle error    }  } |

**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** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Main-function-catch-all early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++ERR51 |  |
| CodeSonar | 8.0p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2023.3 | C++4035, C++4036, C++4037 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Numeric Overflow** | [STD-008-CPP] | Text numeric maximum or minimum limit while incrementing or decrementing value. |

| **Noncompliant Code** |
| --- |
| Incrementing numeric value without testing maximum limit |
| template <typename T>  void test\_overflow()  {  const unsigned long int steps = 5;  const T increment = std::numeric\_limits<T>::max() / steps;  const T start = 0;  T result;  steps += 1;  for (unsigned long int i = 0; i < steps; ++i)  {  result += increment;  }  } |

| **Compliant Code** |
| --- |
| Using maximum limit test to determine if maximum incrementation value exceeds limit to avoid value wrapping. |
| template <typename T>  void test\_overflow()  {  const unsigned long int steps = 5;  const T increment = std::numeric\_limits<T>::max() / steps;  const T start = 0;  T result;  steps += 1;  if (result == std::numeric\_limits<T>::max()) {  std::cout << "OVERFLOW OCCURED!!!" << std::endl;  }  else {  for (unsigned long int i = 0; i < steps; ++i)  {  result += increment;  }  }  } |

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

| **Principles(s):** (1) Validate input data of integer type to ensure integer operations do not wrap around their high and low limits. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | Integer-overflow | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT30 | Implemented |
| CodeSonar | 8.0p0 | ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| Coverity | 2017.07 | INTEGER\_OVERFLOW | Implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Library function overflow** | [STD-009-CPP] | Guarantee that library functions do not overflow |

| **Noncompliant Code** |
| --- |
| The std::copy() algorithm provides no inherent bounds checking and can lead to a buffer overflow. |
| #include <algorithm>  #include <vector>    **void** f(**const** std::vector<**int**> &src) {    std::vector<**int**> dest;    std::copy(src.begin(), src.end(), dest.begin());    // ...  } |

| **Compliant Code** |
| --- |
| The proper way to use std::copy() is to ensure the destination container can hold all the elements being copied to it. |
| #include <algorithm>  #include <vector>  **void** f(**const** std::vector<**int**> &src) {    // Initialize dest with src.size() default-inserted elements    std::vector<**int**> dest(src.size());    std::copy(src.begin(), src.end(), dest.begin());    // ...  } |

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

| **Principles(s):** (1) Architect and Design for Security Policies to ensure library functions behave as expected. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | BADFUNC.BO.\* LANG.MEM.BO LANG.MEM.TBA | A collection of warning classes that report uses of library functions prone to internal buffer overflows. Buffer Overrun Tainted Buffer Access |
| Helix QAC | 2023.3 | DF3526, DF3527, DF3528, DF3529, DF3530, DF3531, DF3532, DF3533, DF3534 |  |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-CTR52-a | Do not pass empty container iterators to std algorithms as destinations |
| Polyspace Bug Finder | R2023b | CERT C++: CTR52-CPP | Checks for library functions overflowing sequence container (rule partially covered). |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Function returning value** | [STD-010-CPP] | Value-returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| No return statement after condition (a < 0) is not met, so not all code paths return a value. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  } |

| **Compliant Code** |
| --- |
| All code paths now return a value. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  **return** a;  } |

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

| **Principles(s):** (1) Architect and Design for Security Policies to ensure functions do not exit abruptly or at unexpected point. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MSC52 |  |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as *function-try-blocks* |
| CodeSonar | 8.0p0 | LANG.STRUCT.MRS | Missing return statement |
| LDRA tool suite | 9.7.1 | 2 D, 36 S | Fully implemented |

### 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.

DevSecOps automation can be introduced into the current DevOps process using CI/CD pipelines such as Jenkins to coordinate between the preproduction and production processes and Docker for container management. Automation can be applied in some phases of the DevOps as follow:

Access and Plan:

* Use DevOps lifecycle tools such as GitLab as Azure DevOps.
* Use identity providers and web access management tools.
* Use two-factor/Multi-factor Authentication tools.
* Secure access points based on role- or attribute-based access control.
* Use privileged account security tools to protect access and API keys.

Design and development:

* Apply and enforce secure coding practices following industry standards such as OWASP and CERT.
* Use static dependency check tools to track dependency vulnerabilities.

Build:

* Use industry-standard tools for securely building and packaging software modules such as MS Build, Maven, Gradle.
* Use trusted open-source code and add additional access control to avoid security breach.
* Use Two-factor/Multifactor Authentication to protect against unauthorized access to code repositories.

Test and Verify:

* Use unit testing tools such as Cucumber, Junit, and Google test.
* Use DAST and IAST tools such as OWASP ZAP, Veracode and Qualys to scan built modules for vulnerabilities.
* Use fuzzing tools such as Radamsa, AFL, and Burp Suite.
* Use vulnerability scanning tools to scan APIs, data access layer, integration layer, and middleware components.

### 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 | Low | Unlikely | Low | P3 | L3 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Likely | High | P9 | L2 |
| STD-009-CPP | High | Likely | Medium | P18 | L1 |
| STD-010-CPP | Medium | Probable | Medium | P8 | L2 |

### 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 | Data at rest should be secured using full disk encryption to prevent against authorized data access in the incident of loss, theft, or device disposal. |
| Encryption at flight | Data in flight should be protected using industry standard cryptographic protocols like the TLS with end-to-end encryption of transmitted data. Use SSH to manage authentication credentials and links. |
| Encryption in use | Use Two-Factor/Multi-Factor Authentication and other identity management tools to ensure authorized data access. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Use secure authentication schemes like multi-factor authentication which requires more than one authentication step to quickly arrest account impersonation attempts to reach privileged and sensitive data. |
| Authorization | Authenticated users must be given only the amount of authority over the system and network that is required for their operations and no more rather than exposing unused access to sensitive resources that could be exploited if such authenticated user was hacked. |
| Accounting | Every activity of authenticated and authorized users must be tracked and audited regularly to be able to quickly notice any suspicious system use or breach and arrest them before they are exploited. Tracking information such as log-on duration, received and sent data, IP addresses, URIs, and other services accessed by an authenticated user must be regularly audited. |

**\***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 | 02/16/2024 | Policy Update | Babatunde Ali-Brown |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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