**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 | This is the process of checking that data entered a system is safe and meets certain criteria before it’s processed. It’s a security measure that helps prevent malicious actors from causing harm to a system. |
| 1. Heed Compiler Warnings | This is the process of listening to your warnings within your compiler. Compilers emit both errors, which prevent your code from compiling at all, and warnings, which indicate a potential problem, but still let your code compile. Using static and dynamic analysis tools to help in eliminating potential security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. If you require different privileges at different times, try dividing the system into distinct intercommunicating subsystems each with an appropriate privilege set. |
| 1. Keep It Simple | This one is straight forward. Keep your software design simple and as small as you can possibly make it. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied, and the protection scheme identifies conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process should be executed with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker must execute arbitrary code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated into an effective quality assurance program. |
| 1. Adopt a Secure Coding Standard | Secure coding standards govern the coding practices, techniques, and decisions that developers make while building software. They aim to ensure that developers write code that minimizes security vulnerabilities. Development tasks can be solved in many ways, with varying levels of complexity. |

### 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] | Do not define a C-style variadic function |

| **Noncompliant Code** |
| --- |
| This function reads an argument until the value of 0 is found. If after the first two iterations the value of 0 is not found it results in an undefined behavior. Then passing any value that is not an INT value also results in an undefined behavior. |
| #include <cstdarg>    **int** add(**int** first, **int** second, ...) {  **int** r = first + second;  **va\_list** va;  **va\_start**(va, second);  **while** (**int** v = **va\_arg**(va, **int**)) {      r += v;    }  **va\_end**(va);  **return** r;  } |

| **Compliant Code** |
| --- |
| A solution to this problem is to expand on the function parameter pack into a list of values as part of a braced initializer list. Since narrowing conversions are not allowed in a braced initializer list, the type of safety is preserved despite the std: : enable\_if not involving any of the variadic arguments. |
| #include <type\_traits>    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg i, Arg j, Ts... all) {  **int** values[] = { j, all... };  **int** r = i;  **for** (auto v : values) {      r += v;    }  **return** r;  } |

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

| **Principles(s):**  3: Architect and Design for Security Policies  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Function-ellipsis | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL50 | [Insert text.] |
| Clang | 3.9 | Cert-dc150-cpp | Check by clang-tidy |
| CodeSonar | 8.1p0 | LANG.STRUCT.ELLIPSIS | Ellipsis |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not depend on the order of evaluation for side effects. |

| **Noncompliant Code** |
| --- |
| In the below code example, i is evaluated more than once in an un-sequenced manner, so the behavior of the expression would become undefined. |
| **void** f(**int** i, **const** **int** \*b) {  **int** a = i + b[++i];    // ...  } |

| **Compliant Code** |
| --- |
| With this solution to the above problem this example is independent of the order of evaluation of the operands and can each be interpreted in only one way. |
| **void** f(**int** i, **const** **int** \*b) {    ++i;  **int** a = i + b[i];    // ...  } |

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

| **Principles(s):**  1: ValidateInput Data  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**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++-EXP50 | [Insert text.] |
| Clang | 3.9 | -Wensequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| CodeSonar | 8.1p0 | LANG.STRUCT.SE.DEC  LANG.STRUCT.SE.INC | Side effects in expression with DEC  Side Effects in expression with INC |
| Éclair | 1.2 | CC2.EXP30 | Fully implemented |

#### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| This example of noncompliant code shows a std::string object created from the result of a call to std::getenv(). Since the std::getenv() returns a null pointer the code can lead to undefined behavior when the variable does not exist. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| The compliant solution to the code above results from the call to std::getenv() are checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    **void** f() {  **const** **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");  **if** (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):** 2: Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Assert\_failure | [Insert text.] |
| CodeSonar | 8.1p0 | LANG.MEM.NPD | Null Pointer Dereference |
| Helix QAX | 2024.2 | DF4770, DF4771, DF4772, DF4773, DF4774 | [Insert text.] |
| Parasoft C/C++ text | 2023.1 | CERT\_CPP-STR51-a | Avoird null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not log unsanitized user input |

| **Noncompliant Code** |
| --- |
| The noncompliant code below logs an untrusted data from an unauthenticated user without data sanitization. |
| **if** (loginSuccessful) {    logger.severe("User login succeeded for: " + username);  } **else** {    logger.severe("User login failed for: " + username);  } |

| **Compliant Code** |
| --- |
| This compliant code below sanitizes the user before logging into to prevent any kind of injection attack. It is done with two parts the login and a dedicated method for sanitizing users. |
| **if** (loginSuccessful) {    logger.severe("User login succeeded for: " + sanitizeUser(username));  } **else** {    logger.severe("User login failed for: " + sanitizeUser(username));  }  **public** String sanitizeUser(String username) {  **return** Pattern.matches("[A-Za-z0-9\_]+", username))        ? username : "unauthorized user";  } |

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

| **Principles(s):**  1: ValidateInput Data  7: Sanitize Data Sent to Other Systems  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| CodeSonar | 8.1p0 | JAVA.IO.Taint.Log | Tainted Log |
| Fortify | [Insert text.] | Log\_Forging | Implemented |
| Klockwork | 2024.1 | SVLOG\_FORGING | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Allocate sufficient memory for an object |

| **Noncompliant Code** |
| --- |
| The noncompliant code below does not allocate the proper space need for a struct tm object because the size of the pointer is being used to determine the size of the pointed-to object. |
| #include <stdlib.h>  #include <time.h>    **struct** **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {  **struct** **tm** \*tmb;    tmb = (**struct** **tm** \*)**malloc**(**sizeof**(tmb));  **if** (tmb == NULL) {  **return** NULL;    }    \*tmb = (**struct** **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };  **return** tmb;  } |

| **Compliant Code** |
| --- |
| The compliant code fixes the problem by allocating more memory for the struct tm object. A simple way to allocate sufficient memory for an object is to pass the pointer type of sizeof. |
| #include <stdlib.h>  #include <time.h>    **struct** **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {  **struct** **tm** \*tmb;    tmb = (**struct** **tm** \*)**malloc**(**sizeof**(\*tmb));  **if** (tmb == NULL) {  **return** NULL;    }    \*tmb = (**struct** **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };  **return** tmb;  } |

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

| **Principles(s):**  2: Heed Compiler Warnings  5: Default Deny  6: Adhere to the Principle of Least Privilege  9: Use Effective Quality Assurance Techniques |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Malloc-size-insufficient | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM35 | [Insert text.] |
| Cppcheck Premium | 24.9.0 | Premium-cert-mem35-c | Partially implemented |
| LDRA tool suite | 9.7.1 | 400 S, 487 S, 115 D | Enhanced enforcement |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory mapped structure that is essential for the code to behave correctly. |
| #include <assert.h>    **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    **int** func(**void**) {  **assert**(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**));  } |

| **Compliant Code** |
| --- |
| To make that code compliant we must use a preprocessor conditional statement to allow the assertion to work properly. |
| **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

| **Principles(s):**  2:Heed Compiler Warnings  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL01 | [Insert text.] |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 8.1p0 | (Customization) | Users can implement a custom check that reports uses of the assert () marco |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| This noncompliant code has no matching handler for the exception thrown for f() nor main() so the std::terminate() is called. |
| **void** throwing\_func() noexcept(**false**);    **void** f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution to the code above the main entry point handles all exceptions which means the stack will be unwound up to the main() function and allows for proper management of external resources. |
| **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):**  9: Use Effective Quality Assurance Techniques  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-allearly-catch-all | Partially checked |
| Axivion Bauhaus suite | 7.2.0 | CartC++-ERR51 | [Insert text.] |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Klocwork | 2024.2 | MISRA.CATCH.ALL | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions and Error Handling | [STD-008-LLL] | Catch exceptions by Ivalue reference. |

| **Noncompliant Code** |
| --- |
| An object of type S is used to initialize the exception object that is later caught by an exception-declaration of type std::exception. The exception-declaration matches the exception object type, so the variable E is copy-initialized from the exception object, resulting in the exception object being sliced. |
| #include <exception>  #include <iostream>    **struct** S : std::exception {  **const** **char** \*what() **const** noexcept override {  **return** "My custom exception";    }  };    **void** f() {  **try** {  **throw** S();    } **catch** (std::exception e) {      std::cout << e.what() << std::endl;    }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the variable declared by the exception-declaration is a lvalue reference. The call to what() results in executing S::what() instead of std::exception::what(). |
| #include <exception>  #include <iostream>    **struct** S : std::exception {  **const** **char** \*what() **const** noexcept override {  **return** "My custom exception";    }  };    **void** f() {  **try** {  **throw** S();    } **catch** (std::exception &e) {      std::cout << e.what() << std::endl;    }  } |

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

| **Principles(s):**  4: Keep It Simple |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Catch-class-by-value | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR61 | [Insert text.] |
| Helix QAC | 2024.2 | C++4031 | [Insert text.] |
| LDRA tool suite | 9.7.1 | 455 S | Fully Implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declarations and Initialization | [STD-009-CPP] | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| two different translation units define a class of the same name with differing definitions. Although the two definitions are functionally equivalent (they both define a class named S with a single, public, nonstatic data member int a), they are not defined using the same sequence of tokens. |
| // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |

| **Compliant Code** |
| --- |
| If the programmer intends for the same class definition to be visible in both translation units because of common usage, the solution is to use a header file to introduce the object into both translation units, as shown in this compliant solution. |
| // S.h  **struct** S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):**  3: Architect and Design for Security Policies  4: Keep It Simple  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 | [Insert text.] |
| Helix QAC | 2024.2 | C++1067, C++1509, C++1510 | [Insert text.] |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Fully Implemented |
| Parasoft C/C++ | 2023.1 | CERT\_CPP-DCL60-a | A class, union or enum name |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-010-CPP] | Do not delete a polymorphic object without a virtual destructor |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, b is a polymorphic pointer type whose static type is Base \* and whose dynamic type is Derived \*. When b is deleted, it results in undefined behavior because Base does not have a virtual destructor. The implicitly declared destructor is not declared as virtual even in the presence of other virtual functions. |
| **struct** Base {  **virtual** **void** f();  };    **struct** Derived : Base {};    **void** f() {    Base \*b = **new** Derived();    // ...  **delete** b;  } |

| **Compliant Code** |
| --- |
| The destructor for Base has an explicitly declared virtual destructor, ensuring that the polymorphic delete operation results in well-defined behavior. |
| **struct** Base {  **virtual** ~Base() = **default**;  **virtual** **void** f();  };    **struct** Derived : Base {};    **void** f() {    Base \*b = **new** Derived();    // ...  **delete** b;  } |

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

| **Principles(s):**  2: Heed Compiler Warnings  10: Adopt a Secure Coding Standard |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Non-virtual-public-destructor-in-non-final-class | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP52 | [Insert text.] |
| Clang | 3.9 | -Wdelete-non-virtual-dtor | [Insert text.] |
| CodeSonar | 8.1p0 | LANG.STRUCT.DNVD | Delete with Non-Virtual Destructor |

### 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.

[Insert your written explanations here.]

### 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 | Probable | Medium | P12 | L1 |
| STD-002-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-005-CPP | High | Probable | High | P6 | L2 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-009-CPP | High | Unlikely | High | P3 | L3 |
| STD-010-CPP | Low | Likely | Low | P9 | 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 at rest | Encryption in rest protects stored data. This may include hard drives, phones, computers, and cloud assets, among others. Protection of this data can be done through encryption tools, disk encryption and security for mobile devices and computers. |
| Encryption in flight | Encryption at flight is about protecting data that is moving. This can be between two devices within a network or moving outside of a network. This can be protected through examples such as email encryption, DLP solutions, and solid network security features, such as firewalls and authentication. It is also important to consider the path data may be taking, and the security of this path. |
| Encryption in use | Encryption in use protects data that is created, edited, or otherwise defined as in-use. Protection of this data can be done by ensuring data control and protection exists prior to use, and in place in the first place. Managing access rights and identity will also minimize risk to this data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the act of confirming one’s identity. This can cover a range of types, but often are examples such as static passwords, one-time passwords, certifications, and biometric credentials. These forms of identification work to ensure a person is who they claim to be. |
| Authorization | Authorization specifies the access rights and privileges of a user, and are an important part of information and computer security. Where authentication confirms an identity, authorization determines what a user can and cannot access in the first place, limiting possible vulnerabilities when someone may interact with sensitive data they may not need to access, or the permissions one may have during access. |
| Accounting | Accounting is the process of keeping track of activity while interacting with a system, showing timestamps, accessed resources, and data transfer information. This is valuable in both creating a “bread crumb trail” in user activity, and also for the purposes of forensic analysis and investigation, should it be required. |

**\***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 | 09/24/2024 | Module 3 | Martin Richardson | [Insert text.] |
| 1.2 | 10/14/2024 | Module 6 | Martin Richardson | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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