**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

# Alex Surprenant

# CS-405

## 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 | Data must be verified before it is input into a system. This will prevent many security vulnerabilities that can arise from lack of, or improper validation of inputs. Attacks like SQL injection and buffer overflow can be prevented with this principle. |
| 1. Heed Compiler Warnings | Compiler warnings should be thoroughly reviewed and addressed when running programs, as they can reveal vulnerabilities in the program or errors that could possibly be exploited. |
| 1. Architect and Design for Security Policies | Security policies should be a part of the beginning of the software development process, in order to be sure that it is considered throughout the lifecycle and thus the project is initiated with security in mind rather than having it added later on in the cycle. |
| 1. Keep It Simple | Creating more complex solutions to problems can more easily lead to bugs and other issues arising in the system, which can create a vulnerability in the system. Solutions should be as simple and maintainable as possible to prevent this. |
| 1. Default Deny | With this principle, unless explicitly permitted, all permission, access and privileges are automatically denied. This way, all threats are automatically blocked and privileges have to be specifically granted to trusted sources |
| 1. Adhere to the Principle of Least Privilege | Similar to default deny, users should only be granted the minimum amount of access they need into a system to perform the task specific to them. This approach will limit the access users have to systems in case of a user’s access becoming compromised. |
| 1. Sanitize Data Sent to Other Systems | Ensuring that data sent to another system has been sanitized will reduce the likelihood of the receiving system becoming compromised if it does not perform the proper input validations on its own. |
| 1. Practice Defense in Depth | Adding layers of security to each level of the system will ensure that there are added protections on each level. This way, in case one layer of security fails at a specific level, there are other layers of security in place to keep the system protected. |
| 1. Use Effective Quality Assurance Techniques | Ensuring effecting QA techniques ties into architect and design for security policies as security testing should be used throughout the development life cycle, in order to expose potential vulnerabilities early on in the lifecycle. QA teams should be testing throughout the lifecycle to find these vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Development teams should create a group of guidelines and standards to adhere to throughout the development lifecycle in order to ensure vulnerabilities are minimized by proper testing, implementation and used the best security practices at their disposal. |

### 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** | **Never qualify a reference type with const or volatile** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | C++ will not allow the value of a reference type to be changed on declaration. All references are effectively const qualified |

| **Noncompliant Code** |
| --- |
| The char is a const-qualified reference. It should be a reference to a const qualified char. |
| **void** f(**char** c) {  **char** &**const** p = c;    p = 'p';    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| The const is removed, and the p is a referenced char |
| **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):** Adopt a Secure Coding Standard  This rule is part of a guideline that is meant to avoid undefined behavior. Developers must follow secure standards that don’t cause undefined behavior |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | 3 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-DCL52** |  |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| Polyspace Bug Finder | R2024b | CERT C++: DCL52-CPP | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| Clang | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not rely on side effects in unevaluated operands** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Relying on an expression to evaluate an operand will not work as the unevaluated operand will not trigger a side effect. Using the operand to evaluate a variable will not function as expected. |

| **Noncompliant Code** |
| --- |
| In this code, b is expected to be the result of a++ inside the sizeof() method. This will not evaluate to the expected output and the result will be b = 14 |
| #include <iostream>  void f() {  int a = 14;  int b = sizeof(a++);  std::cout << a << ", " << b << std::endl;  } |

| **Compliant Code** |
| --- |
| A is incremented outside of the sizeof() operator and the expected value of b is achieved. |
| **void** f() {  **int** a = 14;  **int** b = **sizeof**(a);    ++a;    std::cout << a << ", " << b << std::endl;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard  Unevaluated operands wont execute their side effects in C++, and relying on them to do so can cause unpredictable behavior and vulnerabilities in security. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | 3 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | sizeof | Partially checked |
| CodeSonar | 9.1p0 | LANG.STRUCT.SE.SIZEOF | Side Effects in sizeof |
| LDRA tool suite | 9.7.1 | 54 S, 133 S | Partially implemented |
| Polyspace Bug Finder | R2024b | CERT C++: EXP52-CPP | Checks for logical operator operand with side effects |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | C style strings include a null character at the end of the string, and should be considered when planning for the size of the memory needed for storage. Not doing so can result in buffer overflow. |

| **Noncompliant Code** |
| --- |
| The input to the buff char is unbounded, and can result in buffer overflow. |
| **void** f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| This solution uses a std:string instead of a bounded array. This insures that the data input into the string is not truncated, protecting from buffer overfows. |
| **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):** ValidateInput Data  If the developer fails to allocate enough space for characters and the null terminator, buffer overflows can occur. Using proper sting size will prevent this vulnerability from occurring. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | 9 | 2 |

**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() |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| LDRA tool suite | 9.7.1 | 489 S, 109 D, 66 X, 70 X, 71 X | Partially implemented |
| TrustInSoft Analyzer | 1.38 | mem\_access | Exhaustively verified |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Sanitize data passed to complex subsystems** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C] | String data that is passed to a complex subsystem may contain characters that can be used for SQL injection, and thus should be sanitized before being passed to the system. |

| **Noncompliant Code** |
| --- |
| This code snippet allows an attacker to log into the system by using the USER environmental variable as an argument to the login program. |
| (**void**) execl(LOGIN\_PROGRAM, "login",    "-p",    "-d", slavename,    "-h", host,    "-s", pam\_svc\_name,    (AuthenticatingUser != NULL ? AuthenticatingUser :  **getenv**("USER")),    0); |

| **Compliant Code** |
| --- |
| The addition of the “- -” double dash before the call to getenv(“USER”), which causes getOpt() to stop interpreting options. getOpt() is the function that parses the command line arguments. The call made to execl() is not susceptible to SQL injection because the shell comment interpreter is not envoked. |
| (**void**) execl(LOGIN\_PROGRAM, "login",    "-p",    "-d", slavename,    "-h", host,    "-s", pam\_svc\_name,    "--",    (AuthenticatingUser != NULL ? AuthenticatingUser :  **getenv**("USER")), 0); |

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

| **Principles(s):** Sanitize Data Sent to Other Systems  Sanitizing the data before passing to subsystems will prevent malicious input from being able to take advantage of exploits in the system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | 9 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 |  | Supported by stubbing/taint analysis |
| Coverity | 6.5 | TAINTED\_STRING | Fully implemented |
| LDRA tool suite | 9.7.1 | 108 D, 109 D | Partially implemented |
| Parasoft C/C++test | 2024.2 | CERT\_C-STR02-a CERT\_C-STR02-b CERT\_C-STR02-c | Protect against command injection Protect against file name injection Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Calling a method on deallocated memory can lead to undefined behavior and issues with memory. |

| **Noncompliant Code** |
| --- |
| In this code, the structure S is defined with the member function f(). An instance of S is created with s pointing to an S object. It is then deleted, leaving s as a dangling pointer, which points to deallocated memory, and causing undefined behavior and potential corruption to memory and security flaws. |
| **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In this code snippet, the call to delete s is moved below the last us of the s reference, and thus the function is no longer pointing to deallocated memory, leading to a dangling pointer. |
| **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):** Adopt a Secure Coding Standard  When freed memory is accessed, it can lead to Undefined behavior which can be exploited. Developers should be used to avoid this issue. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | 9 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | dangling\_pointer\_use |  |
| 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 | 9.1p0 | ALLOC.UAF | Use after free |
| Coverity | v7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use a static assertion to test the value of a constant expression** |
| --- | --- | --- |
| **Assertions** | [STD-006-C] | Because assert() calls abort(), it is only valuable for checking incorrect assumptions and not for checking errors at runtime. |

| **Noncompliant Code** |
| --- |
| This code uses assert() to assert memory-mapped property using runtime assertion. Although its use is better than nothing, assert should be used in a function static\_assert(constant-expression, string-literal); |
| **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** |
| --- |
| Using static\_assert() allows the incorrect test assumptions to be evaluated and diagnosed at runtime instead of potentially causing a runtime error. |
| **struct** timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    static\_assert(**sizeof**(**struct** timer) == **sizeof**(unsigned **char**) + **sizeof**(unsigned **int**) + **sizeof**(unsigned **int**),                "Structure must not have any padding"); |

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

| **Principles(s):** Use Effective Quality Assurance Techniques  Using static assertions is a method of QA checking done during compile time. They will detect issues with constant values and prevent errors later in the development cycle. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | 3 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| CodeSonar | 9.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | [Insert text.] |
| ECLAIR | 1.2 | **CC2.DCL03** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Honor exception specifications** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | If a function throws an exception that is not allowed in the exception-specification, it will trigger std::terminate(), which will lead to the program terminating. Functions must only throw exceptions allowed by their exception-specifications. |

| **Noncompliant Code** |
| --- |
| This code snippet might throw an exception as noexcept(), however resize() may throw an exception when it is not possible to allocate requested memory. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {    v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| The following code removes the noexcept specification, which makes it so the function will now allow exceptions. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) {    v.resize(s); // May throw, but that is okay  } |

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

| **Principles(s):** Keep It Simple  Honoring the exceptions will make sure that the functions used in the program will only throw the exceptions that are declared within them, making it easier to understand and maintain the code. Keeping error handling predictable will reduce the risk of unexpected behaviors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | unhandled-throw-noexcept | Partially checked |
| CodeSonar | 9.1p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| LDRA tool suite | 9.7.1 | 56 D | Partially implemented |
| Parasoft C/C++Test | 2024.2 | CERT\_CPP-ERR55-a | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Write constructor member initializers in the canonical order** |
| --- | --- | --- |
| **Object Oriented Programming** | [STD-008-CPP] | Class members and base classes should be initialized in the order they are declared, not the order in the constructors initializer list. Writing the initializers out of order could cause the program to have undefined behavior. |

| **Noncompliant Code** |
| --- |
| In this code snippit, the initializer for C::C() attempts initializing someVal before dependsOnSomeVal to a value that depends on the value of someVal. Because the declaration order does not match the initialization order, an unspecified value will be stored in dependsOnSomeValue. |
| **class** C {  **int** dependsOnSomeVal;  **int** someVal;    **public**:    C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| The ordering of declaration is flipped, and now the constructor initialization is in the correct order, and should have correctly defined behavior. |
| **class** C {  **int** someVal;  **int** dependsOnSomeVal;    **public**:    C(**int** val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

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

| **Principles(s):** Keep It Simple  Writing the member initializers out of order can create unpredictable behavior, and keeping them in the declared order will also help keep the code easy to understand and maintainable. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | initializer-list-order | Fully checked |
| CodeSonar | 9.1p0 | LANG.STRUCT.INIT.OOMI | Out of Order Member Initializers |
| LDRA tool suite | 9.7.1 | 206 S | Fully implemented |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-OOP53-a | List members in an initialization list in the order in which they are declared |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not use std::rand() for generating pseudorandom numbers** |
| --- | --- | --- |
| **Number Generation** | [STD-009-CPP] | The Pseudorandom number generator function from the C++ standard library produces numbers that are not genuinely random. Numbers generated can have a short cycle and can thus be predicted at times. Applications that require true random numbers should use a more sufficient generator. |

| **Noncompliant Code** |
| --- |
| This snippet generates an ID that is predictable and has limited randomness. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    id += std::to\_string(std::**rand**() % 10000);    // ...  } |

| **Compliant Code** |
| --- |
| This compliant snippet utilizes the <random> from C++ to break random number generation into multiple parts. It also uses the Mersenne Twister algorithm as the engine to generate random values, and spreads them across uniform distribution. |
| #include <random>  #include <string>    **void** f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    std::uniform\_int\_distribution<**int**> distribution(0, 10000);    std::random\_device rd;    std::mt19937 engine(rd());    id += std::to\_string(distribution(engine));    // ...  } |

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

| **Principles(s):** Adopt a secure coding standard  The method std::rand() produces predictable values, and can be exploited by a capable malicious actor. Developers utilizing secure coding standards should use stronger and more secure random number generation methods. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | High | 4 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | bad-function (AUTOSAR.26.5.1A) | Fully checked |
| Clang | 4.0 (prerelease) | cert-msc50-cpp | Checked by clang-tidy |
| CodeSonar | 9.1p0 | BADFUNC.RANDOM.RAND | Use of rand |
| ECLAIR | 1.2 | CC2.MSC30 | Fully implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Value-returning functions must return a value from all exit paths** |
| --- | --- | --- |
| **Function Values** | [STD-010-CPP] | A function must return a value if it is declared as a value-returning function. Otherwise it will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| The function does not account for a positive value, and thus not all code paths return a value. |
| **int** absolute\_value(**int** a) {  **if** (a < 0) {  **return** -a;    }  } |

| **Compliant Code** |
| --- |
| The addition of the positive integer return makes it so this function now returns a value for all code paths. |
| **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):** Adopt a secure coding standard  When a function fails to return values on all exit paths, undefined behavior occurs. Developers should utilize secure coding standards which require functions to return a value, creating predictable behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | 8 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | return-implicit | Fully checked |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule, such as *function-try-blocks* |
| CodeSonar | 9.1p0 | LANG.STRUCT.MRS LANG.STRUCT.NVNR | Missing return statement Non-void noreturn, |
| Polyspace Bug Finder | R2024b | CERT C++: MSC52-CPP | Checks for missing return statements (rule partially covered) |

### 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] | Low | Unlikely | High | 3 | 3 |
| [STD-002-CPP] | Low | Unlikely | High | 3 | 3 |
| [STD-003-CPP] | High | Likely | Medium | 9 | 2 |
| [STD-004-C] | High | Likely | Medium | 9 | 2 |
| [STD-005-CPP] | High | Likely | Medium | 9 | 2 |
| [STD-006-C] | Low | Unlikely | High | 3 | 3 |
| [STD-007-CPP] | Low | Likely | Medium | 6 | 2 |
| [STD-008-CPP] | Medium | Unlikely | Medium | 6 | 2 |
| [STD-009-CPP] | Medium | Unlikely | High | 4 | 3 |
| [STD-010-CPP] | Medium | Probable | Medium | 8 | 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 at rest | Data that is currently being held inactively and is not being put to any use. It can be stored in a hard drive, PC or in a database. This type of data is valuable to hackers as it is easy to access. Using data Encryption for storage, as well as access controls such as permissions and authentication, are among the best ways to protect this data. |
| Encryption in flight | Data that is currently being moved from one point to another. Examples would include data moving through email, text messaging, team collaboration tools like Slack and Teams. This type of data can be intercepted and is extremely vulnerable. Developers should use methods such as TLS/SSL to encrypt data in transit, and using checksums to detect corruption in the transmitted data. |
| Encryption in use | Data that is currently being accessed by users via reading, processing or updating. The nature of being directly available makes this type of data extremely vulnerable if left exposed. Developers should use methods such as access controls to restrict users and applications from accessing data that should not be accessed, and continuous monitoring, to detect any unexpected behavior from occurring during data processing. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is how a user proves who they are through providing identification information. This can be through methods such as providing a user name and password, biometric fingerprint or facial recognition. This applies to a secure policy as it is a direct method for a user to confirm identity. |
| Authorization | Involves granting a user the ability to access certain areas of a system, and dictates what actions are allowed to be taken by the user. Authorization can be given by administrators, or by the system depending on the users access privileges. |
| Accounting | This is where the users activity within a system is tracked for accountability purposes. User trends can be analyzed in order to build a picture of usage trends and activity. |

**\***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 | 7/20/25 | Milestone | Alex Surprenant |  |
| 1.2 | 8/14/25 | Final Policy | Alex Surprenant |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

Resources

Djalovic, N. (2025, April 8). *Understanding encryption - data at rest, in motion, in use*. Jatheon Technologies Inc. <https://jatheon.com/blog/data-at-rest-data-in-motion-data-in-use/>

*What is AAA Security?*. Fortinet. (n.d.). https://www.fortinet.com/resources/cyberglossary/aaa-security