**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 | User input and data should be validated and sanitized first before sending to the subsystem or into any data storages. Some methods of sanitation could include removing, replacing, encoding or escaping characters. This will ensure that the input data wont inadvertently expose the program to attacks. |
| 1. Heed Compiler Warnings | You should compile your code with the highest warning levels possible. For any warnings, address them and then recompile your code. Ignoring warning messages can leave your program exposed to attacks or malicious intents. |
| 1. Architect and Design for Security Policies | When designing software always have security at the forefront. You should continually employ the practice of least privilege to users and components. If a system has multiple streams of data, disjoint them and ensure least privilege is enacted on all components to limit attacks and malicious actions. |
| 1. Keep It Simple | By keeping the application design simpler you can maintain the application and ensure that there are no security holes. Larger and overly complex programs can become vulnerable to attacks due to oversight. |
| 1. Default Deny | Access should be based on permission rather than exclusion. The default permission will be denied and only granted access once the appropriate permissions are granted. If exclusion was chosen, you could miss adding a user to the excluded group and would default them to have access causing issues. |
| 1. Adhere to the Principle of Least Privilege | Every task should complete with the least amount of privileges needed. If elevated permissions are required, they should only be held for the duration of the task and then revoked once complete. |
| 1. Sanitize Data Sent to Other Systems | Data that is sent to a subsystem or third party component should be sanitized and cleaned first. The component that receives the data doesn’t know the condition or state that this data came from and doesn’t necessarily sanitize data before use. So it’s the responsibility of the caller to do so. |
| 1. Practice Defense in Depth | By building a defense of multiple levels this will ensure that if one fails, another defense can mitigate attacks and ensure safety. By having multiple levels of security attackers will need to bypass multiple levels in order to achieve what they’re intending to do. |
| 1. Use Effective Quality Assurance Techniques | A solid QA can help identify security issues through a few practices. Through penetration testing and source code audits, QA engineers can help identify security issues and offer assistance in closing these holes. |
| 1. Adopt a Secure Coding Standard | You should adopt and create a secure coding standard for all projects and applications regardless of the language or tools. By adhering to established guidelines and guidelines you create, you can ensure the application will be secure and less prone to attacks. |

### 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** | [DCL-51-CPP] | Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| Common practice is to use a macro in a preprocessor that guards against multiple inclusions of a header file. This is fine, but such a name can clash with reserved names defined in the in the cpp standard template library. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| This solution avoids using leading and trailing underscores of the guard file. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

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

| **Principles(s):** Architect and Design for Security Policies is the policy that maps to this rule. They can be looked at as mapping to one other because it’s important to design your headers and files to adhere to security policies |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wreserved-id-macro  -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not  catch all instances of this rule, such as redefining reserved names. |
| CodeSonar | 7.0p0 | LANG.ID.NU.MK  LANG.STRUCT.DECL.RESERVED | Macro name is C keyword  Declaration of reserved name |
| LDRA tool suite | 9.7.1 | 86 S, 218 S, 219 S, 580 S | The tool suite provides requirements traceability, test management, coding standards compliance, code quality review, code coverage analysis, data-flow and control-flow analysis, unit/integration/target testing, and certification and regulatory support. |
| Polyspace Bug Finder | R2022a | CERT C++: DCL51-CPP | Checks for redefinitions of reserved identifiers (rule partially covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [EXP-50-CPP] | Do no depend on the order of evaluation for side effects |

| **Noncompliant Code** |
| --- |
| The value computation is sequenced before the modification of the operand. |
| // i is modified twice in the same full expression  i = ++i + 1;    // i is read other than to determine the value to be stored  a[i++] = i; |

| **Compliant Code** |
| --- |
| The assignment of i is not modified before the reassignment of i. |
| i = i + 1;  a[i] = i; |

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

| **Principles(s):** Use Effective Quality Assurance Techniques is the principle to map to this rule. This principle can be mapped to this rule because you should be using effective QA techniques to minimize side effects of software. You shouldn’t rely on order of operations and other orders to minimize side effects |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| GCC | 4.9 | -Winvalid-offsetof | Can detect violations of this rule when the -Wsequence-point flag is used |
| SonarQube C/C++ Plugin | 4.10 | IncAndDecMeixedWithOtherOperators | The compiler is generally allowed to remove code that does not have any effect, according to the abstract machine of the C language. This means that if you have a buffer that contains sensitive data (for instance passwords. |
| Coverity | v7.5.0 | EVALUATION\_ORDER | Can detect the specific instance where a statement contains multiple side effects on the same value with an undefined evaluation order because, with different compiler flags or different compilers or platforms, the statement may behave differently |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STR-51-CPP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| The std::string objected is created from the results of the std::getenv() which on failure can return a null pointer. |
| #include <cstdlib>  #include <string>    **void** f() {    std::string tmp(std::**getenv**("TMP"));  **if** (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this solution the results from std::getenv() are checked for null before creating the string |
| #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):** The principle that maps to this rule is ValidateInput Data. You should always validate input data and not create input data from null pointers or null values. All values and pointers should be validated. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | LANG.MEM.NPD | Null pointer dereference |
| Parasoft C/C++test | 2021.2 | Cert\_cpp-str51-A | Avoid null pointer dereferencing |
| Klocwork | 2022.1 | NPD.CHECK.CALL.MIGHT | is a static code analysis tool |
| Helix QAC | 2022.1 | C++4770 | commercial static code analysis software tool |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STR-02-C] | Sanitize data passed to complex subsystems |

| **Noncompliant Code** |
| --- |
| This vulnerability invoked the login program by calling execl(). This call passes unsanitized data from an untrusted source as an argument granting access with heightened permissions. |
| (**void**) execl(LOGIN\_PROGRAM, "login",    "-p",    "-d", slavename,    "-h", host,    "-s", pam\_svc\_name,    (AuthenticatingUser != NULL ? AuthenticatingUser :  **getenv**("USER")),    0); |

| **Compliant Code** |
| --- |
| This solution inserts the “—“ argument before the call to getenv(“USER”) in the call to execl(). This prevents the attacked from adding any additional arguments to the function. |
| (**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):** The principle that can be mapped to this rule is Sanitize Data Sent to Other Systems. You should always sanitize data or input before sending to any subsystem or other system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | IO.INJ.COMMAND  IO.INJ.FMT | Command injection  Format string injection |
| Coverity | 6.5 | TAINTED\_STIRNG | Sanitize data passed to complex subsystems |
| Polyspace Bug Finder | R2022a | CERT C: Rec. STR02-C | Checks for execution of externally controlled command |
| Parasoft C/C++test | 2021.2 | CERT\_C-STR02-a | Protect against command injections |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [MEM-50-CPP] | Do no access freed memory |

| **Noncompliant Code** |
| --- |
| s is dereferenced after it has been deallocated, which can lead to exploitation of arbitrary code ran on the permissions of the vulnerable process |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| The dynamically allocated memory isn’t 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): The principle that adheres to this rule is** Architect and Design for Security Policies. Why this rule and principle map to one other is because you want to engineer and design your code to be secure and if you try to access and use freed memory, this goes against the principle of Architect and Design for Security Policies**.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| 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 free pointer |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy to verify all memory pointers |
| Parasoft Insure++ | 2021.2 | CERT\_CPP-MEM51-c | Properly deallocate dynamically allocated resources |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-MEM50-a | Do no use resources that have been freed |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [DCL-03-C] | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| The usage of the assert() to assert the property concerning the memory-mapped structure should be done within the function and executed |
| #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** |
| --- |
| The use of static\_assert allows for incorrect assumptions to be diagnosed at compile time instead. |
| #include <assert.h>    **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):** By following the principle of Use Effective Quality Assurance Techniques you can follow the rule of using static assertions of constant values. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | Use a static assertion to test the value of a constant expression |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 7.0p0 | Customization | Users can implement a custom check that reports uses of the assert() macro |

#### 

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [ERR-51-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** |
| --- |
| This example, main() handles all exceptions ensuring the handling of any exceptions from the program. |
| **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):** The principle that links to this rule is Heed Compiler Warnings. You shouldn’t ignore any compiler warnings and should always handle all and any exceptions that may stem from their software. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klocwork | 2022.1 | MISRA.CATCH.ALL | Error handling of all exceptions |
| RuleChecker | 20.10 | Main-function-catch-all  Early-catch-all | Error handling for all exceptions of the main |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| CodeSonar | 7.0p0 | LANG.STRUCT.UCTCH | Unreachable Catch |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | [MSC-50-CPP] | Do not use std::rand() for generation pseudorandom numbers |

| **Noncompliant Code** |
| --- |
| The IDs in this code are generated from calling rand(), but these values are predictable and have 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 solution seeds the random number engine ensuring the outputs are truly random and compliant with MSC-51-CPP. |
| #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): The principle that adheres to this rule is Architect and Design for Security Policies. You should design and engineer secure software by following the rule of not using std::rand() for random number generation.** |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | BADFUNC.RANDOM.RAND | Use of rand |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced Enforcement |
| RuleChecker | 20.10 | Bad-function (AUTOSAR.26.5.1A) | Fully Checked |
| PRQA QA-C++ | 4.4 | 5028 | MSC50-CPP. Do not use std::rand() for generating pseudorandom numbers |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Characters and Strings | [STR-53-CPP] | Range check element access |

| **Noncompliant Code** |
| --- |
| The value returned by get\_index() may be greater than the number of elements stored in the string. |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| This example uses std::basic\_string::at() which will throw an error if the pos is out of range |
| #include <stdexcept>  #include <string>  **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");  **try** {      s.at(get\_index()) = '1';    } **catch** (std::out\_of\_range &) {      // Handle error    }  } |

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

| **Principles(s):** Validate Input Data adheres to this rule because we need to validate and ensure that all indexes and ranges are valid and checked. If they’re going to throw an exception, then we should handle it properly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.0p0 | LANG.MEM.BO | Buffer overrun |
| Parasoft C/C++test | 2121.2 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |
| Polyspace Bug Finder | R2022a | CERT C++:STR53-CPP | Checks for: array access out of bounds, array access with tainted index, and pointer dereference with tainted offset |
| Helix QAC | 2022.1 | C++ 3162  C++ 3163  C++ 3164 | STR53-CPP. Range check element access |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | [ERR-50-CPP] | Do no abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| Since f() is registered as an exit handler, it may not handle and close properly for exceptions |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| In this example, f() handles all exceptions thrown by throwing\_func() and wont rethrow |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.  **try** {      throwing\_func();    } **catch** (...) {      // Handle error    }  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

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

| **Principles(s):** I think two principles can belong here: Architect and Design for Security Policies and Keep It Simple. We should handle all exceptions thrown but without terminating the program early. This maps to the policies by keeping it simple in the code base and handling the exceptions and having a solid architecture and design for security. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Stdlib-use | Astrée is a static code analyzer that proves the absence of run­time errors and invalid con­current behavior in safety-critical software written or gen­er­ated in C or C++. |
| CodeSonar | 7.0p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| Polyspace Bug Finder | R2022a | CERT C++: ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered) |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |

### 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.

One area that can be improved in the current DevOps model is to implement a preventing step before the monitor and detect. By adding a prevent step, you can verify data, perform integrity checks, and implement thorough defense in depth measures. Another step would be to include prediction after the maintain and stabilize step. By including a prediction step, you can perform analytics on what to plan for next in the design phase. This will also allow the system to analyze areas that may need more development work as well.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| [EXP-50-CPP] | Medium | Probable | Medium | P8 | L2 |
| [STR-51-CPP] | High | Likely | Medium | P18 | L1 |
| STR-02-C | High | Likely | Medium | P18 | L1 |
| MEM-50-CPP | High | Likely | Medium | P18 | L1 |
| DCL-03-C | Low | Unlikely | High | P1 | L3 |
| ERR-51-CPP | Low | Probable | Medium | P4 | L3 |
| MSC-50-CPP | Medium | Unlikely | Low | P6 | L2 |
| STR-53-CPP | High | Unlikely | Medium | P6 | L2 |
| ERR-50-CPP | Low | Probable | Medium | P4 | L3 |

### 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 | Encrypting data at rest is when data that is not active or being used daily is encrypted by a cryptographic key. Only authorized personnel will have access to these files. This policy applies because we need to ensure that the data that we have saved or stored too needs to be encrypted and made safe. |
| Encryption at flight | To put it simply, this is data that is moving over a network. This data can be easily intercepted and thus should also be encrypted too. This encrypted data should start encrypted and after the handshake between the two applications, the receiver can then decrypt the encrypted data. |
| Encryption in use | In-Use encryption takes a new approach that ensures that sensitive data is never left unsecured, regardless of or lifecycle stage. Data in use could be handled in protected memory or the data can be transformed for use. An example would be the use of a hash of the original data for comparison purposes like when performing password verifications. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication provides a method of identifying a user, typically via password and username. Authentication is important to ensure that our software is secured and only accessed by the correct and authenticated people. If authentication isn’t performed then we can’t ensure who is actually accessing the data. |
| Authorization | Once authentication has been proven, we can implement steps to gain authorization. These steps for instance could be if a user logs in and tries to give commands to the software, the software can confirm or deny these actions based on the users role. So if a user is authorized to complete this task then they can do so. |
| Accounting | Accounting can be measured by the resources a user consumes during network access. This can include the amount of system time or the amount of data sent and received during a session. Accounting is carried out by logging session statistics and usage information. This is important to our policies because we can monitor and log any abnormalities by users. If they are using the network a lot or for prolonged times and something then breaks, we can begin our investigation starting with that person. |

**\***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 | 06/19/2022 | Completed Template | Kalin Mason | Kalin Mason |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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