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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | All input coming from an outside source should be treated as untrusted and be checked to minimize vulnerability. If input data is not validated, it leaves your program open to being attacked and your system breached. It can also lead to SQL injections. The use of syntactical and Semantic strategies is recommended. |
| 1. Heed Compiler Warnings | The use of static and dynamic analysis tools is recommended to mitigate some security flaws. These particular flaws are reported by the compiler. To receive the most accurate and risky flaws, use the highest warning level available for the compiler being used. Then modify the code to fix warnings and flaws. |
| 1. Architect and Design for Security Policies | Design your system in such a way that privileges and security policies are listed separately. The design should have policies only being accessed by those with privileges to those security. Security policies should be implemented in such a way as well as enforced in such a way that any attempts to access them without the correct privileges will cause the user to lose access completely. |
| 1. Keep It Simple | Keeping the design of your application simple allows for mistakes to be less probable. It also allows for other developers, that may work on the code after you, to understand the design with ease. |
| 1. Default Deny | Deny access to all whom do not have permission to access said area of a system. Decisions should be based on if the user has permission to access that area of the system. |
| 1. Adhere to the Principle of Least Privilege | A list of privileges are allocated as necessary. All processes are run with the least the lowest privilege needed to run said process. Higher level privileges are only granted permission for a limited time. |
| 1. Sanitize Data Sent to Other Systems | The removal of dangerous characters from user input that could be passed into complex subsystems. This reduces the risk of injection attacks. Data sanitization should be done before calling any subsystems. |
| 1. Practice Defense in Depth | The implementation of several layers of defense to secure a system. If one layer does not stop an attacker from infiltrating a system, the other layers are in place to do so; or at least minimize the exploitation. |
| 1. Use Effective Quality Assurance Techniques | Fuzz and penetration testing should be done to test a system for flaws and vulnerabilities. Source code audits and security reviews should also be done as well to ensure a system does not consist of any “holes”. |
| 1. Adopt a Secure Coding Standard | Create or adopt a secure coding standard for your system specific to the language used and the platform is will fun on. |

### 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** | **Do not define a C-style variadic function** |
| --- | --- | --- |
| **Data Type** | [STD-50-CPP] | Variadic functions can accept numerous amounts of arguments of different types. These types of defined functions can cause undefined behavior and allow for exploitation of code |

| **Noncompliant Code** |
| --- |
| Example uses a variadic function that gets defined without proper arguments |
| #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** |
| --- |
| Example uses a variadic using a function parameter pack to implement the function. Results in ill-formed function rather than unknown behavior.  Example doesn’t require a recursive expansion of the parameter pack allowing for type safety. |
| #include <type\_traits>    **template** <**typename** Arg, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Arg s) { **return** f + s; }    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Ts... rest) {  **return** f + add(rest...);  } |
| #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):**  *Keep in simple*: This policy for this standard tells developers not to complicate functions and their parameters so that undefine behavior, that may inhibit a system, doesn’t happen.  *Sanitize Data Sent to Other Systems*: Before sending results of the function to another system or calling any subsystems, make sure the data is sanitized to avoid injection attacks.  *Use Effective Quality Assurance Techniques*: Testing the system for these types of functions, that can cause delay or threat to a system, so that they can be caught and fixed to mitigate these possible issues.  *Adopt a Secure Coding Standard*: By adhering to this standard, a developer should avoid using such functions all together. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Artree | 22.10 | Function-ellipsis | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCS50 | N/A |
| Clang | 3.9 | Cert-dcl50-cpp | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **LANG.STRUCT.ELLIPSIS** | Ellipsis |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++2012, C++2625** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2022.3 | **MISRA.FUNC.VARARG** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **41 S** | Fully Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL50-a** | Functions shall not be defined with a variable number of arguments |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl50cpp.html) | Checks for function definition with ellipsis notation (rule fully covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2012, 2625** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **function-ellipsis** | Fully checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**FunctionEllipsis**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-923) |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Do not declare or define a reserved identifier** |
| --- | --- | --- |
| **Data Value** | [STD-51-CPP] | Do not change the definition of a unit that is included in a standard library nor shall use words identical to keywords. Names with underscores are reserved for global namespace. Globally used names and functions should be linked using the same name space. |

| **Noncompliant Code** |
| --- |
| Example uses a macro in a preprocessor conditional with a reserved named in a C++ standard template library.  Example uses a user-defined literal that does not contain the required suffix.  Example contains a potential name clashing from the use of a header that is included within another header  Example has conflicting macros by using conflicting headers |
| #ifndef \_MACRO\_HEADER\_H\_  #define \_MACRO\_HEADER\_H\_  // Contents of <macro\_header.h>  #endif // \_MACRO\_HEADER\_H\_ |
| #include <cstddef>    unsigned **int** operator"" x(**const** **char** \*, std::**size\_t**); |
| #include <cstddef> // std::for size\_t    **static** **const** std::**size\_t** \_max\_limit = 1024;  std::**size\_t** \_limit = 100;    unsigned **int** get\_value(unsigned **int** count) {  **return** count < \_limit ? count : \_limit;  } |
| #include <cinttypes> // for int\_fast16\_t    **void** f(std::int\_fast16\_t val) {  **enum** { MAX\_SIZE = 80 };    // ...  } |

| **Compliant Code** |
| --- |
| Example abstains from using underscores leading or trailing header names in header guard.  Example contains user-defined literal without reserved identifier.  Example uses file scope without underscores  Example refrains from the use of reserved names |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |
| #include <cstddef>    unsigned **int** operator"" \_x(**const** **char** \*, std::**size\_t**); |
| #include <cstddef> // for size\_t    **static** **const** std::**size\_t** max\_limit = 1024;  std::**size\_t** limit = 100;    unsigned **int** get\_value(unsigned **int** count) {  **return** count < limit ? count : limit;  } |
| #include <cinttypes> // for std::int\_fast16\_t    **void** f(std::int\_fast16\_t val) {  **enum** { BufferSize = 80 };    // ...  } |

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

| **Principles(s):**  *Keep It Simple*: Do not complicate the naming of variables. The use of keywords within or already established globally defined variables will only create issues so avoid the use of them.  *Use Effective Quality Assurance Techniques*: Implement tests that will check if variables cause issues and stop a system from running. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **reserved-identifier** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.2p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++5003** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2022.3 | **MISRA.DEFINE.WRONGNAME** **MISRA.DEFINE.WRONGNAME.UNDERSCORE** **MISRA.UNDEF.WRONGNAME** **MISRA.UNDEF.WRONGNAME.UNDERSCORE** **MISRA.STDLIB.WRONGNAME** **MISRA.STDLIB.WRONGNAME.UNDERSCORE** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **86 S, 218 S, 219 S, 580 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL51-a** **CERT\_CPP-DCL51-b** **CERT\_CPP-DCL51-c** **CERT\_CPP-DCL51-d** **CERT\_CPP-DCL51-e** **CERT\_CPP-DCL51-f** | Do not #define or #undef identifiers with names which start with underscore Do not redefine reserved words Do not #define nor #undef identifier 'defined' The names of standard library macros, objects and functions shall not be reused The names of standard library macros, objects and functions shall not be reused (C90) The names of standard library macros, objects and functions shall not be reused (C99) |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl51cpp.html) | Checks for redefinitions of reserved identifiers (rule partially covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **5003** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.22 | [**V1059**](https://pvs-studio.com/en/docs/warnings/v1059/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **reserved-identifier** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**978**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-978) |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Never qualify a reference type with const or volatile** |
| --- | --- | --- |
| **String Correctness** | [STD-52-CPP] | C++ treats all references as type “const”. Can not change value of a reference type. Cv-qualification of a reference type is ignored. Only non-reference type can be cv-qualified. |

| **Noncompliant Code** |
| --- |
| Example uses a const-qualified reference to *char*  Example declares variable correctly but the modification results in a ill-formed program |
| #include <iostream>    **void** f(**char** c) {  **char** &**const** p = c;    p = 'p';    std::cout << c << std::endl;  } |
| #include <iostream>    **void** f(**char** c) {  **const** **char** &p = c;    p = 'p'; // Error: read-only variable is not assignable    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Example omits the use of the keyword “const” |
| #include <iostream>    **void** f(**char** c) {  **char** &p = c;    p = 'p';    std::cout << c << std::endl;  } |

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

| **Principles(s):**  *Keep It Simple*: Do not attempt to change the value of reference data type variables.  *Use Effective Quality Assurance Techniques*: Implement tests that will check if variables cause issues and stop a system from running.  *Head Compiler Warnings*: If there is an attempt to change the value of a reference type, the compile will possibly send an error or warning. Make the changes the compiler suggests.  *Adopt a Secure Coding Standard*: Adopting this standard will prevent compiler or other issues |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL52** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++0014** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.3 | **CERT.DCL.REF\_TYPE.CONST\_OR\_VOLATILE** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **0014** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3708**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3708) |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Do no write syntactically ambiguous declaration** |
| --- | --- | --- |
| **SQL Injection** | [STD-53-CPP] | Make sure syntax is not a *vexing parse* type that would allow for the confusion of if something is declared or is an expression statement |

| **Noncompliant Code** |
| --- |
| Example declares a local variable without the default constructor. Declaration is syntactically ambiguous and can be declared as a function points.  Example shows a vexing parse where the declaration is not parsed correctly |
| #include <iostream>    **struct** Widget {    Widget() { std::cout << "Constructed" << std::endl; }  };    **void** f() {    Widget w();  } |
| #include <iostream>    **struct** Widget {  **explicit** Widget(**int** i) { std::cout << "Widget constructed" << std::endl; }  };    **struct** Gadget {  **explicit** Gadget(Widget wid) { std::cout << "Gadget constructed" << std::endl; }  };    **void** f() {  **int** i = 3;    Gadget g(Widget(i));    std::cout << i << std::endl;  } |

| **Compliant Code** |
| --- |
| Example uses a lock object is given an identifier and converted properly  Example uses 1)A way to ensure syntax to declare a variable instead of a function and 2)A way that uses *braced-init-list* that directly initializes the local variable  Example has complaint way 1) using declaration of a variable with extra set of parentheses surrounding the argument in the constructor call and 2) using a direct initialization of the variable. |
| #include <mutex>    **static** std::mutex m;  **static** **int** shared\_resource;    **void** increment\_by\_42() {    std::unique\_lock<std::mutex> lock(m);    shared\_resource += 42;  } |
| #include <iostream>    **struct** Widget {    Widget() { std::cout << "Constructed" << std::endl; }  };    **void** f() {    Widget w1; // Elide the parentheses    Widget w2{}; // Use direct initialization  } |
| #include <iostream>    **struct** Widget {  **explicit** Widget(**int** i) { std::cout << "Widget constructed" << std::endl; }  };    **struct** Gadget {  **explicit** Gadget(Widget wid) { std::cout << "Gadget constructed" << std::endl; }  };  **void** f() {  **int** i = 3;    Gadget g1((Widget(i))); // Use extra parentheses    Gadget g2{Widget(i)}; // Use direct initialization    std::cout << i << std::endl;  } |

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

| **Principles(s):**  *Validate Input Data*: Any input coming in should be not trusted and sanitized. Use syntactical strategies that are without obscurity in their naming convention.  *Heed Compiler Warnings*: Use these warnings to improve the security of the code. Modifying code to resolve compiler warnings will lessen vulnerabilities and help a program run more smoothly; makes a for a quicker system.  *Architect and Design for Security Policies*: Be sure to only allow for data to be passed or collected by those with the appropriate privileges. Reject any whom attempt to access data or areas that they do not have privileges in.  *Keep It Simple*: Do not complicate the naming of variables nor make them too simplistic. Variable naming should be meaningful and simple.  *Default Deny*: All decisions whether or not a user can access and area of the system or not should be solely based on whether or not they possess the level of privilege is takes to access said area.  *Adhere to the Principle of Least Privileges*: Be sure to allocate different levels of privileges for different areas of a system and processes. If a user does not have the proper privilege to access and area, reject them. If they do, only allow access for a limited time.  *Sanitize Data Sent to Other Systems*: Make sure there are no hazardous characters within the data before calling any subsystems.  *Use Effective Quality Assurance Techniques*: Implement tests that will check if variables or injection attempts that can cause issues and stop a system from running.  *Head Compiler Warnings*: If there is an attempt to change the value of a reference type, the compile will possibly send an error or warning. Make the changes the compiler suggests.  *Practice Defense in Depth*: By implementing the principles above, Several layers are in place to minimize any exploitations of a system. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | **P2** | **L3** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **LANG.STRUCT.DECL.FNEST** | Nested Function Declaration |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++1109, C++2510** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.3 | **CERT.DCL.AMBIGUOUS\_DECL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **296 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL53-a** **CERT\_CPP-DCL53-b CERT\_CPP-DCL53-c** | Parameter names in function declarations should not be enclosed in parentheses Local variable names in variable declarations should not be enclosed in parentheses Avoid function declarations that are syntactically ambiguous |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl53cpp.html) | Checks for declarations that can be confused between:   * Function and object declaration * Unnamed object or function parameter declaration   Rule fully covered. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2502, 2510** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wvexing-parse |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3468**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3468) |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Overload allocation and deallocation functions as a pair in the same scope** |
| --- | --- | --- |
| **Memory Protection** | [STD-54-CPP] | Overload allocation and deallocation paired in same scope. By not overloading a dynamic storage function, a violation of deallocation rule. This can result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Example has an allocation function that’s overloaded but a deallocation function is not declared  Example has one function overloaded at the class scope and another that is not |
| #include <Windows.h>  #include <new>    **void** \*operator **new**(std::**size\_t** size) noexcept(**false**) {  **static** **HANDLE** h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  **if** (h) {  **return** ::HeapAlloc(h, 0, size);    }  **throw** std::bad\_alloc();  }    // No corresponding global delete operator defined. |
| #include <new>    **extern** "C++" **void** update\_bookkeeping(**void** \*allocated\_ptr, std::**size\_t** size, **bool** alloc);    **struct** S {  **void** \*operator **new**(std::**size\_t** size) noexcept(**false**) {  **void** \*ptr = ::operator **new**(size);      update\_bookkeeping(ptr, size, **true**);  **return** ptr;    }  }; |

| **Compliant Code** |
| --- |
| Example has an allocation function with a deallocation function at the global scope.  Example corrects the scope of the second function |
| #include <Windows.h>  #include <new>    **class** HeapAllocator {  **static** **HANDLE** h;  **static** **bool** init;    **public**:  **static** **void** \*alloc(std::**size\_t** size) noexcept(**false**) {  **if** (!init) {        h = ::HeapCreate(0, 0, 0); // Private, expandable heap.        init = **true**;      }    **if** (h) {  **return** ::HeapAlloc(h, 0, size);      }  **throw** std::bad\_alloc();    }    **static** **void** dealloc(**void** \*ptr) noexcept {  **if** (h) {        (**void**)::HeapFree(h, 0, ptr);      }    }  };    **HANDLE** HeapAllocator::h = nullptr;  **bool** HeapAllocator::init = **false**;    **void** \*operator **new**(std::**size\_t** size) noexcept(**false**) {  **return** HeapAllocator::alloc(size);  }    **void** operator **delete**(**void** \*ptr) noexcept {  **return** HeapAllocator::dealloc(ptr);  }  public:  static void \*alloc(std::size\_t szT) noexcept(false) {  if (!init) {  handle = ::HeapCreate(0, 0, 0); // Private, expandable heap.  } init = true;  if (handle) {  } return ::HeapAlloc(handle, 0, szT);  }  static void dealloc(void \*ptr) noexcept {  if (handle) {  };} } (void)::HeapFree(h, 0, ptr);  HANDLE HeapAllocator::handle = nullptr;  bool HeapAllocator::init = false;  void \*operator new(std::size\_t szT) noexcept(false) {  return HeapAllocator::alloc(szT);  }  void operator delete(void \*ptr) noexcept {  } return HeapAllocator::dealloc(ptr); |
| #include <new>    **extern** "C++" **void** update\_bookkeeping(**void** \*allocated\_ptr, std::**size\_t** size, **bool** alloc);    **struct** S {  **void** \*operator **new**(std::**size\_t** size) noexcept(**false**) {  **void** \*ptr = ::operator **new**(size);      update\_bookkeeping(ptr, size, **true**);  **return** ptr;    }    **void** operator **delete**(**void** \*ptr, std::**size\_t** size) noexcept {      ::operator **delete**(ptr);      update\_bookkeeping(ptr, size, **false**);    }  }; |

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

| **Principles(s):**  *Head Compiler Warnings*: Any warnings from the compiler should be taken seriously as to make sure the system does not crash or behave unexpectedly.  *Keep It Simple*: If memory needs to be allocated, it is only natural that it would need to be deallocated eventually. Knowing that the allocation of memory must be overloaded is important as it violates the deallocation rule. Use comments to ensure subsequent developers do not move the deallocation outside of the allocation scope and are sure to overload the allocation of memory.  *Use Effective Quality Assurance Techniques*: Test to make sure that allocation and deallocation happens withing the same scope and memory allocations are overloaded. Use console outputs to confirm or deny that these things have taken place properly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **new-delete-pairwise** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL54** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | misc-new-delete-overloads | Checked with clang-tidy. |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++2160** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.3 | **CERT.DCL.SAME\_SCOPE\_ALLOC\_DEALLOC** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL54-a** | Always provide new and delete together |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL54-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl54cpp.html) | Checks for mismatch between overloaded operator new and operator delete (rule fully covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2160** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **new-delete-pairwise** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S1265**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-1265) |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Avoid cycles during initialization of static objects** |
| --- | --- | --- |
| **Assertions** | [STD-55-CPP] | Since padding bits can be present within a class object instance, and pointers from one domain to another can pass sensitive data, one must make sure that such data is not contained within the padded bits. |

| **Noncompliant Code** |
| --- |
| Example uses a kernel space, copying data from the user space arg.  Example directly initializes a value-initialized arg that ends up being proceeded with a zero and makes the padding bits initialized with a zero.  Example runs a kernel space to copy data from the user space arg but padding bits contain sensitive data. |
| #include <cstddef>    **struct** test {  **int** a;  **char** b;  **int** c;  };    // Safely copy bytes to user space  **extern** **int** copy\_to\_user(**void** \*dest, **void** \*src, std::**size\_t** size);    **void** do\_stuff(**void** \*usr\_buf) {    test arg{1, 2, 3};    copy\_to\_user(usr\_buf, &arg, **sizeof**(arg));  } |
| #include <cstddef>    **struct** test {  **int** a;  **char** b;  **int** c;  };    // Safely copy bytes to user space  **extern** **int** copy\_to\_user(**void** \*dest, **void** \*src, std::**size\_t** size);    **void** do\_stuff(**void** \*usr\_buf) {    test arg{};      arg.a = 1;    arg.b = 2;    arg.c = 3;      copy\_to\_user(usr\_buf, &arg, **sizeof**(arg));  } |
| #include <cstddef>    **class** base {  **public**:  **virtual** ~base() = **default**;  };    **class** test : **public** **virtual** base {    alignas(32) **double** h;  **char** i;    unsigned j : 80;  **protected**:    unsigned k;    unsigned l : 4;    unsigned **short** m : 3;  **public**:  **char** n;  **double** o;      test(**double** h, **char** i, unsigned j, unsigned k, unsigned l, unsigned **short** m,  **char** n, **double** o) :      h(h), i(i), j(j), k(k), l(l), m(m), n(n), o(o) {}    **virtual** **void** foo();  };    // Safely copy bytes to user space.  **extern** **int** copy\_to\_user(**void** \*dest, **void** \*src, std::**size\_t** size);    **void** do\_stuff(**void** \*usr\_buf) {    test arg{0.0, 1, 2, 3, 4, 5, 6, 7.0};    copy\_to\_user(usr\_buf, &arg, **sizeof**(arg));  } |

| **Compliant Code** |
| --- |
| Example serializes the data then copies to untrusted context.  Example uses explicitly declared padding bits within the structure.  Example serializes the object’s data then copies data to untrusted context. |
| #include <cstddef>  #include <cstring>    **struct** test {  **int** a;  **char** b;  **int** c;  };    // Safely copy bytes to user space.  **extern** **int** copy\_to\_user(**void** \*dest, **void** \*src, std::**size\_t** size);    **void** do\_stuff(**void** \*usr\_buf) {    test arg{1, 2, 3};    // May be larger than strictly needed.    unsigned **char** buf[**sizeof**(arg)];    std::**size\_t** offset = 0;      std::**memcpy**(buf + offset, &arg.a, **sizeof**(arg.a));    offset += **sizeof**(arg.a);    std::**memcpy**(buf + offset, &arg.b, **sizeof**(arg.b));    offset += **sizeof**(arg.b);    std::**memcpy**(buf + offset, &arg.c, **sizeof**(arg.c));    offset += **sizeof**(arg.c);      copy\_to\_user(usr\_buf, buf, offset /\* size of info copied \*/);  } |
| #include <cstddef>    **struct** test {  **int** a;  **char** b;  **char** padding\_1, padding\_2, padding\_3;  **int** c;      test(**int** a, **char** b, **int** c) : a(a), b(b),      padding\_1(0), padding\_2(0), padding\_3(0),      c(c) {}  };  // Ensure c is the next byte after the last padding byte.  static\_assert(offsetof(test, c) == offsetof(test, padding\_3) + 1,                "Object contains intermediate padding");  // Ensure there is no trailing padding.  static\_assert(**sizeof**(test) == offsetof(test, c) + **sizeof**(**int**),                "Object contains trailing padding");        // Safely copy bytes to user space.  **extern** **int** copy\_to\_user(**void** \*dest, **void** \*src, std::**size\_t** size);    **void** do\_stuff(**void** \*usr\_buf) {    test arg{1, 2, 3};    copy\_to\_user(usr\_buf, &arg, **sizeof**(arg));  } |
| #include <cstddef>  #include <cstring>    **class** base {  **public**:  **virtual** ~base() = **default**;  };  **class** test : **public** **virtual** base {    alignas(32) **double** h;  **char** i;    unsigned j : 80;  **protected**:    unsigned k;    unsigned l : 4;    unsigned **short** m : 3;  **public**:  **char** n;  **double** o;      test(**double** h, **char** i, unsigned j, unsigned k, unsigned l, unsigned **short** m,  **char** n, **double** o) :      h(h), i(i), j(j), k(k), l(l), m(m), n(n), o(o) {}    **virtual** **void** foo();  **bool** serialize(unsigned **char** \*buffer, std::**size\_t** &size) {  **if** (size < **sizeof**(test)) {  **return** **false**;      }        std::**size\_t** offset = 0;      std::**memcpy**(buffer + offset, &h, **sizeof**(h));      offset += **sizeof**(h);      std::**memcpy**(buffer + offset, &i, **sizeof**(i));      offset += **sizeof**(i);      unsigned loc\_j = j; // Only sizeof(unsigned) bits are valid, so this is not narrowing.      std::**memcpy**(buffer + offset, &loc\_j, **sizeof**(loc\_j));      offset += **sizeof**(loc\_j);      std::**memcpy**(buffer + offset, &k, **sizeof**(k));      offset += **sizeof**(k);      unsigned **char** loc\_l = l & 0b1111;      std::**memcpy**(buffer + offset, &loc\_l, **sizeof**(loc\_l));      offset += **sizeof**(loc\_l);      unsigned **short** loc\_m = m & 0b111;      std::**memcpy**(buffer + offset, &loc\_m, **sizeof**(loc\_m));      offset += **sizeof**(loc\_m);      std::**memcpy**(buffer + offset, &n, **sizeof**(n));      offset += **sizeof**(n);      std::**memcpy**(buffer + offset, &o, **sizeof**(o));      offset += **sizeof**(o);        size -= offset;  **return** **true**;    }  };    // Safely copy bytes to user space.  **extern** **int** copy\_to\_user(**void** \*dest, **void** \*src, **size\_t** size);    **void** do\_stuff(**void** \*usr\_buf) {    test arg{0.0, 1, 2, 3, 4, 5, 6, 7.0};      // May be larger than strictly needed, will be updated by    // calling serialize() to the size of the buffer remaining.    std::**size\_t** size = **sizeof**(arg);    unsigned **char** buf[**sizeof**(arg)];  **if** (arg.serialize(buf, size)) {      copy\_to\_user(usr\_buf, buf, **sizeof**(test) - size);    } **else** {      // Handle error    }  } |

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

| **Principles(s):**  *Validate Input Data*: Make sure that the input is syntactically correct and semantically written. Make sure there are no underflow or overflow bounds being violated.  *Sanitize Data Send to Other Systems*: Instantiating objects often requires placing data into the instance through arguments. The data being passing into the instance needs to be sanitized before being passed in. Checking for SQL injection characters is very important here.  *Adopt a Secure Coding Standard*: [STD-54-CPP] – Memory Protection should be implemented here so that sensitive data doesn’t pass into padded bits of an instance of an object. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL55** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.2p0 | **MISC.PADDING.POTB** | Padding Passed Across a Trust Boundary |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++4941, C++4942, C++4943** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL55-a** | A pointer to a structure should not be passed to a function that can copy data to the user space |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL55-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl55cpp.html) | Checks for information leakage due to structure padding (rule partially covered) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Avoid cycles during initialization of static objects** |
| --- | --- | --- |
| **Declarations and Initializations** | [STD-56-CPP] | During a static variable declaration initialization, functions should not be reenters. In doing so, the behavior of a program is undefined. |

| **Noncompliant Code** |
| --- |
| Example uses a factorial function using caching during a recursion initialization of a static local array cache causing undefined behavior  Example uses a value from one file in another but when initialized causes a “static initialization order fiasco” causing behavior to be unspecified |
| #include <stdexcept>    **int** fact(**int** i) noexcept(**false**) {  **if** (i < 0) {      // Negative factorials are undefined.  **throw** std::domain\_error("i must be >= 0");    }    **static** **const** **int** cache[] = {      fact(0), fact(1), fact(2), fact(3), fact(4), fact(5),      fact(6), fact(7), fact(8), fact(9), fact(10), fact(11),      fact(12), fact(13), fact(14), fact(15), fact(16)    };    **if** (i < (**sizeof**(cache) / **sizeof**(**int**))) {  **return** cache[i];    }    **return** i > 0 ? i \* fact(i - 1) : 1;  } |
| // file.h  #ifndef FILE\_H  #define FILE\_H    **class** Car {  **int** numWheels;    **public**:    Car() : numWheels(4) {}  **explicit** Car(**int** numWheels) : numWheels(numWheels) {}    **int** get\_num\_wheels() **const** { **return** numWheels; }  };  #endif // FILE\_H    // file1.cpp  #include "file.h"  #include <iostream>    **extern** Car c;  **int** numWheels = c.get\_num\_wheels();    **int** main() {    std::cout << numWheels << std::endl;  }    // file2.cpp  #include "file.h"    Car get\_default\_car() { **return** Car(6); }  Car c = get\_default\_car(); |
| // Assume that this class is provided by a 3rd party and it is not something  // that can be modified by the user.  **class** Bad {    ~Bad() noexcept(**false**);  }; |
| **class** SomeClass {    Bad bad\_member;  **public**:    ~SomeClass()  **try** {      // ...    } **catch**(...) {      // Handle the exception thrown from the Bad destructor.    }  }; |

| **Compliant Code** |
| --- |
| Example uses a zero-initialization to decide what member of an array has been given a value. Then returns cached value.  Example uses “construct on first use” that fixes static initialization order issue. |
| #include <stdexcept>    **int** fact(**int** i) noexcept(**false**) {  **if** (i < 0) {      // Negative factorials are undefined.  **throw** std::domain\_error("i must be >= 0");    }    **static** **const** **int** cache[] = {      fact(0), fact(1), fact(2), fact(3), fact(4), fact(5),      fact(6), fact(7), fact(8), fact(9), fact(10), fact(11),      fact(12), fact(13), fact(14), fact(15), fact(16)    };    **if** (i < (**sizeof**(cache) / **sizeof**(**int**))) {  **return** cache[i];    }    **return** i > 0 ? i \* fact(i - 1) : 1;  } |
| // file.h  #ifndef FILE\_H  #define FILE\_H    **class** Car {  **int** numWheels;    **public**:    Car() : numWheels(4) {}  **explicit** Car(**int** numWheels) : numWheels(numWheels) {}    **int** get\_num\_wheels() **const** { **return** numWheels; }  };  #endif // FILE\_H    // file1.cpp  #include "file.h"  #include <iostream>    **int** &get\_num\_wheels() {  **extern** Car c;  **static** **int** numWheels = c.get\_num\_wheels();  **return** numWheels;  }    **int** main() {    std::cout << get\_num\_wheels() << std::endl;  }    // file2.cpp  #include "file.h"    Car get\_default\_car() { **return** Car(6); }  Car c = get\_default\_car(); |

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

| **Principles(s):**  *Validate Input Data*: Any input coming in should be not trusted and sanitized. Make sure that the input is syntactically correct and semantically written. Make sure there are no underflow or overflow bounds being violated. Use syntactical strategies that are without obscurity in their naming convention.  *Head Compiler Warnings*: Any warnings from the compiler should be taken seriously as to make sure the system does not crash or behave unexpectedly. Use these warnings to improve the security of the code.  *Keep in simple*: This policy tells developers not to complicate functions and their parameters so that undefine behavior, that may inhibit a system, doesn’t happen.  *Sanitize Data Sent to Other Systems*: Before sending results of the function to another system or calling any subsystems, make sure the data is sanitized to avoid injection attacks or hacks from corrupt data.  *Use Effective Quality Assurance Techniques*: Testing the output of the functions will show if they are executing correctly. Uses of console outputs to confirm that these exceptions have taken place properly or not.  *Practice Defense in Depth*: By implementing the principles above, Several layers are in place to minimize any exploitations of a system. Exceptions should only execute after the code has been checked and then attempted to be implemented. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | **P2** | **L3** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.2p0 | **LANG.STRUCT.INIT.CYCLE**  **LANG.STRUCT.INIT.UNORDERED** | Initialization Cycle  Unordered Initialization |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++1552, C++1554, C++1704** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **6 D** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL56-a** | Avoid initialization order problems across translation units by replacing non-local static objects with local static objects |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL56-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl56cpp.html) | Checks for:   * Recursive initialization of static variables * Undetermined initialization order of global variables   Rule fully covered. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not let exceptions escape from destructors or deallocation functions** |
| --- | --- | --- |
| **Exceptions** | [STD-57-CPP] | Sometimes a terminating a destructor can cause undefined behavior. Instead of terminating by throwing an exception, use *noexcept(true)* implicitly or *noecept* explicitly. |

| **Noncompliant Code** |
| --- |
| Example may throw an exception even if it was called as a result of an exception being thrown  Example solves the termination issue but may keep the destructor from releasing resources because of its workaround of normal destructor processes  Example rethrows exception because control reaches the end of handler in a try-block function of a constructor or destructor.  Example results in undefined behavior when an exception is thrown from a deallocation function |
| #include <stdexcept>    **class** S {  **bool** has\_error() **const**;    **public**:    ~S() noexcept(**false**) {      // Normal processing  **if** (has\_error()) {  **throw** std::logic\_error("Something bad");      }    }  }; |
| #include <stdexcept>    **class** S {  **bool** has\_error() **const**;    **public**:    ~S() noexcept(**false**) {      // Normal processing  **if** (has\_error()) {  **throw** std::logic\_error("Something bad");      }    }  }; |
| // Assume that this class is provided by a 3rd party and it is not something  // that can be modified by the user.  **class** Bad {    ~Bad() noexcept(**false**);  }; |
| **class** SomeClass {    Bad bad\_member;  **public**:    ~SomeClass()  **try** {      // ...    } **catch**(...) {      // Handle the exception thrown from the Bad destructor.    }  }; |

| **Compliant Code** |
| --- |
| Example has destructors perform same way with or without an active exception.  Example doesn’t throw an exception if deallocation fails. |
| **class** SomeClass {    Bad bad\_member;  **public**:    ~SomeClass()  **try** {      // ...    } **catch**(...) {      // Catch exceptions thrown from noncompliant destructors of      // member objects or base class subobjects.        // NOTE: Flowing off the end of a destructor function-try-block causes      // the caught exception to be implicitly rethrown, but an explicit      // return statement will prevent that from happening.  **return**;    }  }; |
| **class** SomeClass {    Bad bad\_member;  **public**:    ~SomeClass()  **try** {      // ...    } **catch**(...) {      // Catch exceptions thrown from noncompliant destructors of      // member objects or base class subobjects.        // NOTE: Flowing off the end of a destructor function-try-block causes      // the caught exception to be implicitly rethrown, but an explicit      // return statement will prevent that from happening.  **return**;    }  }; |
| #include <cstdlib>  #include <stdexcept>    **bool** perform\_dealloc(**void** \*);  **void** log\_failure(**const** **char** \*);    **void** operator **delete**(**void** \*ptr) noexcept(**true**) {  **if** (perform\_dealloc(ptr)) {      log\_failure("Deallocation of pointer failed");      std::**exit**(1); // Fail, but still call destructors    }  } |

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

| **Principles(s):**  *Validate Input Data*: Any input coming in should be not trusted and sanitized. Make sure that the input is syntactically correct and semantically written. Make sure there are no underflow or overflow bounds being violated. Use syntactical strategies that are without obscurity in their naming convention.  *Head Compiler Warnings*: Any warnings from the compiler should be taken seriously as to make sure the system does not crash or behave unexpectedly. Use these warnings to improve the security of the code. Modifying code to resolve compiler warnings will lessen vulnerabilities and help a program run more smoothly; makes a for a quicker system.  *Architect and Design for Security Policies*: Be sure to only allow for data to be passed or collected by those with the appropriate privileges. Reject any whom attempt to access data or areas that they do not have privileges in. A developer could use *noexcept(true)* to implement this policy.  *Keep in simple*: This policy tells developers not to complicate functions and their parameters so that undefine behavior, that may inhibit a system, doesn’t happen. Exceptions should be used with the same simplistic technique to ensure that what is happening within it is clear and understandable.  *Adhere to the Principle of Least Privileges*: Be sure to allocate different levels of privileges for different areas of a system and processes. If a user does not have the proper privilege to access and area, reject them. If they do, only allow access for a limited time. A developer could use *noexcept(true)* to implement this policy.  *Sanitize Data Sent to Other Systems*: Before sending results of the function to another system or calling any subsystems, make sure the data is sanitized to avoid injection attacks. *Throw()*s should be used to reject a user instead of exceptions  *Use Effective Quality Assurance Techniques*: Testing the execution of the exceptions will show if they are executing correctly. Uses of console outputs to confirm that these exceptions have taken place properly or not.  *Practice Defense in Depth*: By implementing the principles above, Several layers are in place to minimize any exploitations of a system. Exceptions should only execute after the code has been checked and then attempted to be implemented.  *Adopt a Secure Coding Standard*:  [STD-50-CPP] – Data Type  [STD-51-CPP] – Data Value  [STD-52-CPP] – String Correctness  : Use these Coding Standards to establish that the correct data type is being used with the correct data value, and string correctness where applicable, so that exceptions are executed correctly.  [STD-54-CPP] – Memory Protection : Exceptions should adhere to this Coding Standard throughout their implementation as to not incidentally pass sensitive data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **destructor-without-noexcept delete-without-noexcept** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL57** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.2p0 | **LANG.STRUCT.EXCP.CATCH**  **LANG.STRUCT.EXCP.THROW** | Use of catch  Use of throw |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++2045, C++2047, C++4032, C++4631** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Klocwork) | 2022.3 | **MISRA.DTOR.THROW** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **453 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL57-a** **CERT\_CPP-DCL57-b** | Never allow an exception to be thrown from a destructor, deallocation, and swap Always catch exceptions |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL57-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl57cpp.html) | Checks for class destructors exiting with an exception (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.22 | [**V509**](https://pvs-studio.com/en/docs/warnings/v509/)**,** [**V1045**](https://pvs-studio.com/en/docs/warnings/v1045/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **destructor-without-noexcept** **delete-without-noexcept** | Fully checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Do not modify the standard namespaces** |
| --- | --- | --- |
| [Student Choice] | [DCL-58-CPP] | Namespaces make it harder to have conflicting identifiers from other regions. Declaring new namespaces in std namespace can cause undefined behavior. |

| **Noncompliant Code** |
| --- |
| Example declares x by adding it to the namespace resulting in undefined behavior  Example adds a template specialization into the std namespace and results in undefined behavior. |
| **namespace** std {  **int** x;  } |
| #include <functional>  #include <iostream>  #include <string>    **class** MyString {    std::string data;    **public**:    MyString(**const** std::string &data) : data(data) {}    **const** std::string &get\_data() **const** { **return** data; }  };    **namespace** std {  **template** <>  **struct** plus<string> : binary\_function<string, MyString, string> {    string operator()(**const** string &lhs, **const** MyString &rhs) **const** {  **return** lhs + rhs.get\_data();    }  };  }    **void** f() {    std::string s1("My String");    MyString s2(" + Your String");    std::plus<std::string> p;      std::cout << p(s1, s2) << std::endl;  } |

| **Compliant Code** |
| --- |
| Example places the declaration of a variable within the namespace std without a reserved name.  Example defines a new std derivative so that a different std can be used without needing any modifications to the std namespace  Example invokes a constructor that converts constructed class from std |
| **namespace** nonstd {  **int** x;  } |
| #include <functional>  #include <iostream>  #include <string>    **class** MyString {    std::string data;    **public**:    MyString(**const** std::string &data) : data(data) {}    **const** std::string &get\_data() **const** { **return** data; }  };    **struct** my\_plus : std::binary\_function<std::string, MyString, std::string> {    std::string operator()(**const** std::string &lhs, **const** MyString &rhs) **const** {  **return** lhs + rhs.get\_data();    }  };    **void** f() {    std::string s1("My String");    MyString s2(" + Your String");    my\_plus p;      std::cout << p(s1, s2) << std::endl;  } |
| #include <functional>  #include <iostream>  #include <string>    **class** MyString {    std::string data;    **public**:    MyString(**const** std::string &data) : data(data) {}    **const** std::string &get\_data() **const** { **return** data; }  };    **namespace** std {  **template** <>  **struct** plus<MyString> {    MyString operator()(**const** MyString &lhs, **const** MyString &rhs) **const** {  **return** lhs.get\_data() + rhs.get\_data();    }  };  }    **void** f() {    std::string s1("My String");    MyString s2(" + Your String");    std::plus<MyString> p;      std::cout << p(s1, s2).get\_data() << std::endl;  } |

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

| **Principles(s):**  *Head Compiler Warnings*: Any warnings from the compiler should be taken seriously as to make sure the system does not crash or behave unexpectedly. Use these warnings to improve the security of the code.  *Architect and Design for Security Policies*: Be sure to only allow for data to be passed or collected by those with the appropriate privileges. Reject any whom attempt to access data or areas that they do not have privileges in. Any incoming data that attempts to impose a different namespace into a systems code should be rejected.  *Keep in simple*: This policy tells developers not to complicate functions and their parameters so that undefine behavior, that may inhibit a system, doesn’t happen. A developer should only use a single namespace to not complicate the code and create undefined behavior.  *Sanitize Data Sent to Other Systems*: Before sending results of the function to another system or calling any subsystems, make sure the data is sanitized to avoid injection attacks. *Throw()*s should be used to reject a user instead of exceptions.  *Use Effective Quality Assurance Techniques*: Testing the execution of the code will show if they are executing correctly. Uses of console outputs to confirm that these exceptions have taken place properly or not. Testing should be done to make sure that only one namespace is used.  *Practice Defense in Depth*: By implementing the principles above, Several layers are in place to minimize any exploitations of a system. Exceptions should only execute after the code has been checked and then attempted to be implemented.  *Adopt a Secure Coding Standard*:  [STD-50-CPP] – Data Type  [STD-51-CPP] – Data Value  [STD-52-CPP] – String Correctness  : Use these Coding Standards to establish that the correct data type is being used with the correct data value, and string correctness where applicable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL58** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.2p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++3180, C++3181, C++3182** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.3 | **CERT.DCL.STD\_NS\_MODIFIED** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **4032, 4035, 4631** |  |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio) | 7.22 | [**V1061**](https://pvs-studio.com/en/docs/warnings/v1061/) |  |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [**S3470**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-3470) |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Obey the one-definition rule** |
| --- | --- | --- |
| [Student Choice] | [STD-60-CPP] | Be sure that linking behaves deterministically so that each program has one definition of every non-inline function. |

| **Noncompliant Code** |
| --- |
| Example uses two different translations of units in a defined class and a struct named the same.  Example uses a implementation-defined *#pragma* translation unit in one class causing different layouts in each and violates the ODR  Example uses the definition of a function with out using identical translation units. |
| // a.cpp  **struct** S {  **int** a;  };    // b.cpp  **class** S {  **public**:  **int** a;  }; |
| // s.h  **struct** S {  **char** c;  **int** a;  };    **void** init\_s(S &s);    // s.cpp  #include "s.h"    **void** init\_s(S &s); {    s.c = 'a';    s.a = 12;  }    // a.cpp  #pragma pack(push, 1)  #include "s.h"  #pragma pack(pop)    **void** f() {    S s;    init\_s(s);  } |
| **const** **int** n = 42;    **int** g(**const** **int** &lhs, **const** **int** &rhs);    **inline** **int** f(**int** k) {  **return** g(k, n);  } |

| **Compliant Code** |
| --- |
| Example includes header files of both sources of the same named variable  Example uses unique definitions for each of the classes  Example removes the implementation-defined structure member  Example must change the n variable within the function to a not odr-used one, declares said variable to have an external linkage, or not use an inline definition of the function.  Example replaces the n variable with an enumerator. |
| // S.h  **struct** S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |
| // a.cpp  **namespace** {  **struct** S {  **int** a;  };  }    // b.cpp  **namespace** {  **class** S {  **public**:  **int** a;  };  } |
| // s.h  **struct** S {  **char** c;  **int** a;  };    **void** init\_s(S &s);    // s.cpp  #include "s.h"    **void** init\_s(S &s); {    s.c = 'a';    s.a = 12;  }    // a.cpp  #include "s.h"    **void** f() {    S s;    init\_s(s);  } |
| **const** **int** n = 42;    **int** g(**int** lhs, **int** rhs);    **inline** **int** f(**int** k) {  **return** g(k, n);  } |
| **enum** Constants {    N = 42  };    **int** g(**const** **int** &lhs, **const** **int** &rhs);    **inline** **int** f(**int** k) {  **return** g(k, N);  } |

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

| **Principles(s):**  *Head Compiler Warnings*: Any warnings from the compiler should be taken seriously as to make sure the system does not crash or behave unexpectedly. Use these warnings to improve the security of the code. Modifying code to resolve compiler warnings will lessen vulnerabilities and help mitigate undefined behavior that could occur by not heading this standard.  *Keep in simple*: This policy tells developers not to complicate functions and their parameters so that undefine behavior, that may inhibit a system, doesn’t happen. Use unique names for each variable that makes sense to what its purpose it.  *Adhere to the Principle of Least Privileges*: Be sure to allocate different levels of privileges for different areas of a system and processes. This prevents a malicious user from infiltrating a system and making changes to the code that would result in undefined behavior or exposure to vulnerabilities.  *Use Effective Quality Assurance Techniques*: Test to make sure that names are not repeated and/or that the functions operate as expected.  *Adopt a Secure Coding Standard*:  [STD-50-CPP] – Data Type  [STD-51-CPP] – Data Value  [STD-52-CPP] – String Correctness  : Use these Coding Standards to establish that the correct data type is being used with the correct data value, and string correctness where applicable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL60** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.2p0 | **LANG.STRUCT.DEF.FDH LANG.STRUCT.DEF.ODH** | Function defined in header file Object defined in header file |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.3 | **C++1067, C++1509, C++1510** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **286 S, 287 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.1 | **CERT\_CPP-DCL60-a** | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2022b | [CERT C++: DCL60-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl60cpp.html) | Checks for inline constraints not respected (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **type-compatibility** **definition-duplicate** **undefined-extern** **undefined-extern-pure-virtual** **external-file-spreading** **type-file-spreading** | Partially checked |

### 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.

**Pre-Production Process:**

The addition of Digital.Ai, formally XebiaLabs, product *Release* would automate the *Assess and Plan* and *Design* stages. This tool is to be used as a workflow manager to standardize and automate, eventually fully automated, these stages and improve efficiency. Pre-built planning, managing, and design templates should be implemented to increase efficiency with prioritizing, collaboration, and tracking with its visualization single view component. This tool also can automate processes of release and deployment without having to script, as well as preemptively assess for risk throughout changes made. This will create more reliable and effective audits. Automated audits should be performed

A *Git* tool, *Github* or *BitBucket* for example, shall be integrated into the *Design* and *Build* stages. Here the use of repositories will ensure that Coding Standards STD-[50-52, 56, and 57]-CPP, as well as some principles and best practices, are applied throughout the code. This tool will also allow for faster collaboration that’ll catch any discrepancies throughout the code and grants developers to a chance to fix issues that may come up, especially during the build, while still maintaining the original and other versions of the code that can be revisited without starting from scratch. Automation of code being pushed should be done several times a day.

**Post-Production Process:**

*Deploy*, by Digital.Ai, should be utilized at the *Transition and Health Check*, *Monitor and Detect*, and *Respond* stages of the *Post-Production* process. This tool uses automation to reuse dynamic rules for deployment, implements security protocols audit logs and role-based access controls, enforces dependency management, preform analysis, generate reports, and anticipates risk and automates rollbacks if it detects a chance of failure to the system. Furthermore, this tool uses optimized deployment configurations, secrets management, and parameterized configurations. This will enforce STD-[53-55, and 57]-CPP Coding Standards as it would implement each of them for the purpose of security. The frequency of this tool’s utilization should be completed at least daily; at additional points when a new dependency or component is implemented, issues are reported, changing service providers, and when a significant change to a version is made; before its deployment.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-50-CPP | High | Probable | Medium | **P12** | **L1** |
| STD-51-CPP | Low | Unlikely | Low | **P3** | **L3** |
| STD-52-CPP | Low | Unlikely | Low | **P3** | **L3** |
| STD-53-CPP | Low | Unlikely | Medium | **P2** | **L3** |
| STD-54-CPP | Low | Probable | Low | **P6** | **L2** |
| STD-55-CPP | Low | Unlikely | High | **P1** | **L3** |
| STD-56-CPP | Low | Unlikely | Medium | **P2** | **L3** |
| STD-57-CPP | Low | Likely | Medium | **P6** | **L2** |
| STD-58-CPP | High | Unlikely | Medium | **P6** | **L2** |
| STD-60-CPP | High | Unlikely | High | **P3** | **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 | A defense that protects a system from attack when housing code on a disk/computer storage. A developer should create encryption algorithms any time they are storing data there. This will prevent an attacker from infiltrating a system that is not live. |
| Encryption at flight | A defense that protects a system from attack when transmitting data over a network or internet. Implement this by using the correct data transfer protocol, such as SCP, that will safely transfer data and using TLS on web servers with all https connections. |
| Encryption in use | A defense that protects a system from attack when data is actively being used. As it is a substantial vulnerability, data in use should be encrypted to mitigate an attack taking place on stored data that is in the memory. Encryption can be done by either using homomorphic encryption, that qualifies encrypted data as plain text, or with searchable symmetric encryption, that gives users segments of relevant encrypted data with the absence of a completely decrypted dataset. This should be put into uses as long as that data is in use; it lives in the memory until not used. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process of making sure a user is who they state that they are. It requires each user to have a unique username and password so that their credentials can be checked for legitimacy. This policy is used to prevent malicious actors from entering a system. This policy is used with user logins and the addition of new users. |
| Authorization | This policy employs the least privileges policy in that it assigns a different level to each of its users. Certain levels will allow access to certain areas of a system. This is used to limit the amount of access points into the inner system. In a system, this policy is used to grant user level access to valid users or grant admin-level access to an administrator to make changes to the database. |
| Accounting | Here a record is kept, with the use of an external server, for the logging statistics and session duration of each user. This can be used, by an administrator, to check what files have been accessed by the user. |

**\***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 |  |
| 1.2 | 12/16/22 | Record the meanings of the 10 security principles and Coding Standards; label, name, and rationalize the standard, demonstrate examples and explanations of noncompliant and compliant code. | Danielle Monroe | [Insert text.] |
| 1.3 | 12/20/2022 | Added principles, threat, level, and tools to each of the Coding Standards, chose tools to be used with the said Coding Standard, suggested tools to be implemented within the DevOps process that’ll enforce the standard and where they should be imposed, and filled out a summary of risk assessment for each. Policies for the three types of encryption and each element of the Triple-A framework were created by explaining what each was, how it is to be used as well as why and when it applies. Lastly, a map of the principles what documented by listing the name of the principle with its description and implication. | Danielle Monroe | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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