**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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 10](#_Toc52464062)

[Coding Standard 5 12](#_Toc52464063)

[Coding Standard 6 15](#_Toc52464064)

[Coding Standard 7 17](#_Toc52464065)

[Coding Standard 8 19](#_Toc52464066)

[Coding Standard 9 21](#_Toc52464067)

[Coding Standard 10 23](#_Toc52464068)

[Defense-in-Depth Illustration 25](#_Toc52464069)

[Project One 25](#_Toc52464070)

[1. Revise the C/C++ Standards 25](#_Toc52464071)

[2. Risk Assessment 25](#_Toc52464072)

[3. Automated Detection 25](#_Toc52464073)

[4. Automation 25](#_Toc52464074)

[5. Summary of Risk Assessments 26](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 26](#_Toc52464076)

[7. Map the Principles 28](#_Toc52464077)

[Audit Controls and Management 29](#_Toc52464078)

[Enforcement 29](#_Toc52464079)

[Exceptions Process 29](#_Toc52464080)

[Distribution 30](#_Toc52464081)

[Policy Change Control 30](#_Toc52464082)

[Policy Version History 30](#_Toc52464083)

[Appendix A Lookups 30](#_Toc52464084)

[Approved C/C++ Language Acronyms 30](#_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 | Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability). Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files |
| 1. Heed Compiler Warnings | Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set. |
| 1. Keep It Simple | Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystem such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a [security flaw](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-securityflaw) from becoming an exploitable vulnerability and/or limit the consequences of a successful [exploit](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-exploit). For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment |
| 1. Use Effective Quality Assurance Techniques | Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

### 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** | [Include the appropriate type information in function declarators](https://wiki.sei.cmu.edu/confluence/display/c/DCL07-C.+Include+the+appropriate+type+information+in+function+declarators) |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Function declarators must be declared with the appropriate type information, including a return type and parameter list. If type information is not properly specified in a function declarator, the compiler cannot properly check function type information. When using standard library calls, the easiest (and preferred) way to obtain function declarators with appropriate type information is to include the appropriate header file. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses the identifier-list form for parameter declarations: |
| **int** max(a, b)  **int** a, b;  {  **return** a > b ? a : b;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, int is the type specifier, max(int a, int b) is the function declarator, and the block within the curly braces is the function body: |
| **int** max(**int** a, **int** b) {  **return** a > b ? a : b;  } |

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

| **Principles(s):** Valid input data- Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability). Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files |
| --- |

**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=87152428) | 23.04 | **function-prototype**  **implicit-function-declaration** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.FUNCS.PROT LANG.STRUCT.DECL.IMPT** | Incomplete function prototype Implicit Type |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL07** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **21 S** **135 S** **170 S** | Fully implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | [Do not begin integer constants with 0 when specifying a decimal value](https://wiki.sei.cmu.edu/confluence/display/c/DCL18-C.+Do+not+begin+integer+constants+with+0+when+specifying+a+decimal+value) |
| --- | --- | --- |
| **Data Value** | [STD-002-C] | The C Standard defines octal constants as a 0 followed by octal digits (0 1 2 3 4 5 6 7). Programming errors can occur when decimal values are mistakenly specified as octal constants. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a decimal constant is mistakenly prefaced with zeros so that all the constants are a fixed length: |
| i\_array[0] = 2719;  i\_array[1] = 4435;  i\_array[2] = 0042; |

| **Compliant Code** |
| --- |
| To avoid using wrong values and to make the code more readable, do not preface constants with zeroes if the value is meant to be decimal: |
| i\_array[0] = 2719;  i\_array[1] = 4435;  i\_array[2] =   42; |

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

| **Principles(s):** Keep It Simple- don’t add unnecessary zeros before a whole number otherwise it views them as a decimal and can cause issues which is what Keep It Simple references. |
| --- |

**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=87152428) | 23.04 | **octal-constant** | Fully Checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **LANG.TYPE.OC** | Octal constant |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **83 S** | Fully Implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2022.2 | **CERT\_C-DCL18-a** **CERT\_C-DCL18-b** | Octal and hexadecimal escape sequences shall be terminated Octal constants (other than zero) shall not be used |

#### Coding Standard 3

| **Coding Standard** | **Label** | [Do not attempt to modify string literals](https://wiki.sei.cmu.edu/confluence/display/c/STR30-C.+Do+not+attempt+to+modify+string+literals) |
| --- | --- | --- |
| **String Correctness** | [STD-003-C] | At compile time, string literals are used to create an array of static storage duration of sufficient length to contain the character sequence and a terminating null character. String literals are usually referred to by a pointer to (or array of) characters. Ideally, they should be assigned only to pointers to (or arrays of) const char or const wchar\_t. It is unspecified whether these arrays of string literals are distinct from each other. The behavior is [undefined](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior) if a program attempts to modify any portion of a string literal. Modifying a string literal frequently results in an access violation because string literals are typically stored in read-only memory. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the char pointer str is initialized to the address of a string literal. Attempting to modify the string literal is [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior): |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| As an array initializer, a string literal specifies the initial values of characters in an array as well as the size of the array. This code creates a copy of the string literal in the space allocated to the character array str. The string stored in str can be modified safely. |
| **char** str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** Heed Compiler Warnings- A warning may be given when trying to access a string literally and obtaining an access violation because of it. Watching for these warning s will inform the developer to modify the correctness of the strings being used. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **string-literal-modfication** **write-to-string-literal** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-STR30** | Fully implemented |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **PW** | Deprecates conversion from a string literal to "char \*" |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **157 S** | Partially implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | [Create files with appropriate access permissions](https://wiki.sei.cmu.edu/confluence/display/c/FIO06-C.+Create+files+with+appropriate+access+permissions) |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C] | Creating a file with insufficiently restrictive access permissions may allow an unprivileged user to access that file. Although access permissions are heavily dependent on the file system, many file-creation functions provide mechanisms to set (or at least influence) access permissions. When these functions are used to create files, appropriate access permissions should be specified to prevent unintended access. |

| **Noncompliant Code** |
| --- |
| The fopen() function does not allow the programmer to explicitly specify file access permissions. In this noncompliant code example, if the call to fopen() creates a new file, the access permissions are [implementation-defined](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-implementation-definedbehavior): |
| **char** \*file\_name;  **FILE** \*fp;    /\* Initialize file\_name \*/    fp = **fopen**(file\_name, "w");  **if** (!fp){    /\* Handle error \*/  } |

| **Compliant Code** |
| --- |
| The u character can be thought of as standing for "umask," meaning that these are the same permissions that the file would have been created with had it been created by fopen(). In this compliant solution, the u mode character is omitted so that the file is opened with restricted privileges (regardless of the umask): |
| **char** \*file\_name;  **FILE** \*fp;    /\* Initialize file\_name \*/    errno\_t res = fopen\_s(&fp, file\_name, "wx");  **if** (res != 0) {    /\* Handle error \*/  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems, Validate Input Data, Default Deny – both of these principles apply in preventing an SQL injection from taking place. Validate Input Data talks about making sure the input data is validated to prevent an SQL injection from taking place. The Default Deny is at any point the system notices there may have been an attempt at an SQL injection then all access is denied. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **customization** | CodeSonar's custom checking infrastructure allows users to implement checks such as the following.   * A check for all uses of fopen(). * A check for calls to open() with only two arguments. * A check for calls to open() where the third argument does not satisfy some specified requirement. |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **C5013** | [Insert text.] |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Enhanced Enforcement |
| There were only 3 tools listed. | There were only 3 tools listed. | There were only 3 tools listed. | There were only 3 tools listed. |

#### Coding Standard 5

| **Coding Standard** | **Label** | [Do not access freed memory](https://wiki.sei.cmu.edu/confluence/display/c/MEM30-C.+Do+not+access+freed+memory) |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C] | Evaluating a pointer—including dereferencing the pointer, using it as an operand of an arithmetic operation, type casting it, and using it as the right-hand side of an assignment—into memory that has been deallocated by a memory management function is [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior). Pointers to memory that has been deallocated are called dangling pointers. Accessing a dangling pointer can result in exploitable [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability).  According to the C Standard, using the value of a pointer that refers to space deallocated by a call to the free() or realloc() function is undefined behavior. (See [undefined behavior 177](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior).)  Reading a pointer to deallocated memory is undefined behavior because the pointer value is [indeterminate](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-indeterminatevalue) and might be a [trap representation](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-traprepresentation). Fetching a trap representation might perform a hardware trap (but is not required to).  It is at the memory manager's discretion when to reallocate or recycle the freed memory. When memory is freed, all pointers into it become invalid, and its contents might either be returned to the operating system, making the freed space inaccessible, or remain intact and accessible. As a result, the data at the freed location can appear to be valid but change unexpectedly. Consequently, memory must not be written to or read from once it is freed. |

| **Noncompliant Code** |
| --- |
| This example from Brian Kernighan and Dennis Ritchie [[Kernighan 1988](https://wiki.sei.cmu.edu/confluence/display/c/AA.+Bibliography#AA.Bibliography-Kernighan88)] shows both the incorrect and correct techniques for freeing the memory associated with a linked list. In their (intentionally) incorrect example, p is freed before p->next is executed, so that p->next reads memory that has already been freed. |
| #include <stdlib.h>    **struct** node {  **int** value;  **struct** node \*next;  };    **void** free\_list(**struct** node \*head) {  **for** (**struct** node \*p = head; p != NULL; p = p->next) {  **free**(p);    }  } |

| **Compliant Code** |
| --- |
| Kernighan and Ritchie correct this error by storing a reference to p->next  in q before freeing p: |
| #include <stdlib.h>    **struct** node {  **int** value;  **struct** node \*next;  };    **void** free\_list(**struct** node \*head) {  **struct** node \*q;  **for** (**struct** node \*p = head; p != NULL; p = q) {      q = p->next;  **free**(p);    }  } |

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

| **Principles(s):** Keep it Simple and Practice Defense in Depth- it’s important to not leave dangling pointers available to access freed memory. Keep it Simple and also Defense in Depth are both principles to protect against this. If a pointer is dereferenced then it should be removed, but making sure to remove it correctly is key. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **dangling\_pointer\_use** | Supported  Astrée reports all accesses to freed allocated memory. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-MEM30** | Detects memory accesses after its deallocation and double memory deallocations |
| |  |  | | --- | --- | | [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) |  | | 7.3p0 | **ALLOC.UAF** | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | |  |  | | --- | --- | | **USE\_AFTER\_FREE** |  | | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 6

| **Coding Standard** | **Label** | [Use a static assertion to test the value of a constant expression](https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression) |
| --- | --- | --- |
| **Assertions** | [STD-006-C] | Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability) (see [MSC11-C. Incorporate diagnostic tests using assertions](https://wiki.sei.cmu.edu/confluence/display/c/MSC11-C.+Incorporate+diagnostic+tests+using+assertions)). The runtime assert() macro has some limitations, however, in that it incurs a runtime overhead and because it calls abort(). Consequently, the runtime assert() macro is useful only for identifying incorrect assumptions and not for runtime error checking. As a result, runtime assertions are generally unsuitable for server programs or embedded systems. |

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

| **Compliant Code** |
| --- |
| This portable compliant solution uses static\_assert: |
| #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):** Use Effective Quality Assurance Techniques, **Defense in Depth- using tools and measures to create layers of defense in order to prevent a security flaw from becoming a vulnerability.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully Implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | [Honor exception specifications](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR55-CPP.+Honor+exception+specifications) |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | If a function throws an exception other than one allowed by its exception-specification, it can lead to an [implementation-defined](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-implementation-definedbehavior) termination of the program ([except.spec], paragraph 9).  If a function declared with a dynamic-exception-specification throws an exception of a type that would not match the exception-specification, the function std::unexpected() is called. The behavior of this function can be overridden but, by default, causes an exception of std::bad\_exception to be thrown. Unless std::bad\_exception is listed in the exception-specification, the function std::terminate() will be called.  Similarly, if a function declared with a noexcept-specification throws an exception of a type that would cause the noexcept-specification to evaluate to false, the function std::terminate() will be called.  Calling std::terminate() leads to implementation-defined termination of the program. To prevent [abnormal termination](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination) of the program, any function that declares an exception-specification should restrict itself, as well as any functions it calls, to throwing only allowed exceptions. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a function is declared as nonthrowing, but it is possible for std::vector::resize() to throw an exception when the requested memory cannot be allocated. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {    v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the function's noexcept-specification is removed, signifying that the function allows all exceptions. |
| #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) {    v.resize(s); // May throw, but that is okay  } |

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

| **Principles(s):** Heed Compiler Warnings and Keep it Simple – Keeping in unnecessary exceptions when the program is released is bad practice. Exceptions should only be used in early development and only when the developer is looking for an exception and knows exactly how to handle one. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **unhandled-throw-noexcept** | Partially checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | |  | | --- | | **LANG.STRUCT.EXCP.THROW** | | Use of Throw |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **56 D** | Partially implemented |
| [Parasoft C/C++Test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-ERR55-a** | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |

#### Coding Standard 8

| **Coding Standard** | **Label** | [Detect and handle memory allocation errors](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM52-CPP.+Detect+and+handle+memory+allocation+errors) |
| --- | --- | --- |
| Memory Protection | [STD-008-CPP] | The default memory allocation operator, ::operator new(std::size\_t), throws a std::bad\_alloc exception if the allocation fails. Therefore, you need not check whether calling ::operator new(std::size\_t) results in nullptr. The nonthrowing form, ::operator new(std::size\_t, const std::nothrow\_t &), does not throw an exception if the allocation fails but instead returns nullptr. The same behaviors apply for the operator new[] versions of both allocation functions. Additionally, the default allocator object (std::allocator) uses ::operator new(std::size\_t) to perform allocations and should be treated similarly. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, an array of int is created using ::operator new[](std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. Because ::operator new[](std::size\_t) can throw an exception if the allocation fails, it could lead to [abnormal termination](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination) of the program. |
| #include <cstring>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** **int**[size];    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error  **return**;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

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

| **Principles(s):** Keep it Simple and Defense in Depth- The throwing of exceptions when it comes to memory needs to be thoroughly looked into. Keep it simple prevents these exceptions from falling through the cracks as well as defense in depth to find these issues before they become vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Coverity) | 7.5 | **CHECKED\_RETURN** | Finds inconsistencies in how function call return values are handled |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **45 D** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2022.2 | **CERT\_CPP-MEM52-a** **CERT\_CPP-MEM52-b** | Check the return value of new Do not allocate resources in function argument list because the order of evaluation of a function's parameters is undefined |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: MEM52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcmem52cpp.html) | Checks for unprotected dynamic memory allocation (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | [Ensure that operations on signed integers do not result in overflow](https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow) |
| --- | --- | --- |
| **Data Value** | [STD-009-C] | Signed integer overflow is [undefined behavior 36](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior). Consequently, [implementations](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-implementation) have considerable latitude in how they deal with signed integer overflow. (See [MSC15-C. Do not depend on undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/MSC15-C.+Do+not+depend+on+undefined+behavior).) An implementation that defines signed integer types as being modulo, for example, need not detect integer overflow. Implementations may also trap on signed arithmetic overflows, or simply assume that overflows will never happen and generate object code accordingly.  It is also possible for the same conforming implementation to emit code that exhibits different behavior in different contexts. For example, an implementation may determine that a signed integer loop control variable declared in a local scope cannot overflow and may emit efficient code on the basis of that determination, while the same implementation may determine that a global variable used in a similar context will wrap. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in a signed integer overflow during the addition of the signed operands si\_a and si\_b: |
| **void** func(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum = si\_a + si\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution ensures that the addition operation cannot overflow, regardless of representation: |
| #include <limits.h>    **void** f(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum;  **if** (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||        ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {      /\* Handle error \*/    } **else** {      sum = si\_a + si\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** Validate input Data, Keep it simple, Defense in depth – all of these check to make sure that all integers have a set min and max on them to prevent overflow. Keep it simple acknowledges that when the team is writing code this should always be on the forefront of their mind for secure coding. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **integer-overflow** | Fully checked |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **TAINTED\_SCALAR**  **BAD\_SHIFT** | Implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **C2800, C2860**  **C++2800, C++2860**  **DF2801, DF2802, DF2803, DF2861, DF2862, DF2863** | Nothing is listed. |

#### Coding Standard 10

| **Coding Standard** | **Label** | [Cast characters to unsigned char before converting to larger integer sizes](https://wiki.sei.cmu.edu/confluence/display/c/STR34-C.+Cast+characters+to+unsigned+char+before+converting+to+larger+integer+sizes) |
| --- | --- | --- |
| **String Correctness** | [STD-010-C] | Signed character data must be converted to unsigned char before being assigned or converted to a larger signed type. This rule applies to both signed char and (plain) char characters on implementations where char is defined to have the same range, representation, and behaviors as signed char.  However, this rule is applicable only in cases where the character data may contain values that can be interpreted as negative numbers. For example, if the char type is represented by a two's complement 8-bit value, any character value greater than +127 is interpreted as a negative value. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example is taken from a [vulnerability](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability) in bash versions 1.14.6 and earlier that led to the release of CERT Advisory [CA-1996-22](http://www.cert.org/advisories/CA-1996-22.html). This vulnerability resulted from the sign extension of character data referenced by the c\_str pointer in the yy\_string\_get() function in the parse.y module of the bash source code: |
| **static** **int** yy\_string\_get(**void**) {  **register** **char** \*c\_str;  **register** **int** c;      c\_str = bash\_input.location.string;    c = EOF;      /\* If the string doesn't exist or is empty, EOF found \*/  **if** (c\_str && \*c\_str) {      c = \*c\_str++;      bash\_input.location.string = c\_str;    }  **return** (c);  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the result of the expression \*c\_str++ is cast to unsigned char before assignment to the int variable c: |
| **static** **int** yy\_string\_get(**void**) {  **register** **char** \*c\_str;  **register** **int** c;      c\_str = bash\_input.location.string;    c = EOF;      /\* If the string doesn't exist or is empty, EOF found \*/  **if** (c\_str && \*c\_str) {      /\* Cast to unsigned type \*/      c = (unsigned **char**)\*c\_str++;        bash\_input.location.string = c\_str;    }  **return** (c);  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques, Defense in depth for testing and finding these mistakes, and Keep it Simple to hopefully prevent this from happening in the first place by applying secure coding techniques consistently while developing a program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **char-sign-conversion** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-STR34** | Fully implemented |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.3p0 | **MISC.NEGCHAR** | Negative Character Value |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.STR34** | Fully implemented |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

When transitioning from DevOps to DevSecOps, the main thing to remember is that now everyone has a security responsibility. That means that Development, IT operations and Application Delivery all have a duty to uphold the highest level of security while bringing a product to life. Initiating the DevSecOps process would lie between Design and Build. We need to design a product to be built with the best coding standards as outlined in the policy. The idea is that we start thinking about the implementation of these standards before starting to build and when we start to build, is when we fully implement the SecDevOps process. During the Design process, we can use OWASP as outlined in the diagram above. While building, we will seek out a trusted repository such as Azure DevOps. We will use designated tools as outlined in this security document and plug-in’s from our specified IDE’s. Security testing will be implemented the same way, but more in depth as everyone has a role now in upholding a level of security to look for new vulnerabilities as they arise.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Low | Low | 1 |
| STD-002-C | Low | Unlikely | Low | Low | 1 |
| STD-003-C | Low | Likely | Low | Medium | 3 |
| STD-004-C | Medium | Probable | High | Medium | 4 |
| STD-005-C | High | Likely | Medium | High | 5 |
| STD-006-C | Low | Unlikely | High | Low | 2 |
| STD-007-CPP | Low | Likely | Low | Medium | 3 |
| STD-008-CPP | High | Likely | Medium | High | 5 |
| STD-009-C | High | Likely | High | High | 5 |
| STD-010-C | Medium | Probable | Medium | Medium | 4 |

### 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 | Encryption at rest is used to protect data that is stored on a disk. This means any data that is stored needs to be encrypted that is stored at the server level and database. Any encrypter can be used for this such as Advanced Encryption Standard (AES). |
| Encryption at flight | Encryption at flight is any data that is moving over a network. This can be emails, messages or data sent from one computer to another etc. When using cloud services we will use keys to encrypt and decrypt data that is passed over a network to prevent data loss. Additionally, password access will be utilized to access data that has been sent over a network and received to add another layer of security. |
| Encryption in use | Encryption in use allows us to encrypt underlying data as well as analyze all data requests immediately in real time to catch and block suspicious requests. This can be when files are open, or real time access to a database, that poses the most risk. Multi Factor Authentication (MFA) can be utilized to increase security. For the data in use encryption, we can use Secure Encrypted Virtualization (SEV). |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is how we identify a user. This is typically done using a valid username and password that has been set up as their user login prior to access being granted. Each user has a unique identification combination that allows verifies their authentication, including addition of new users, who will be set up by an admin account to create a new username and password. |
| Authorization | After a user has logged in successfully through Authentication, the user now has certain areas in which they have been authorized to have access too. This can be command input, or files available to them. Typically, after authentication is granted the user immediately logs in within their specified user level of access granted by authorization. |
| Accounting | This is how we monitor a user’s activity while they have been granted access. This includes the time, amount of data a user has received or sent during this session, changes to the database and the files the user accessed. This information is very useful for statistical reasons as well as security monitoring as well. |

**\***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 |  |
| 2.0 | 05/18/2023 | Security Principles and Coding Standards updated | Breunna Bingham | Breunna Bingham |
| 3.0 | 6/9/2023 | Updated: Coding Standards, Risk Assessment, Automation, Summery of Risk Assessments, Encryption Policies (rest, flight and use), Triple -A Framework, and Mapping Principles to each standard. | Breunna Bingham | Breunna Bingham |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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