**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 | Validating input allows the application to compare the input with the expected input. If they are a match, access is granted. If they are not a match, the input is considered unsuitable and the data cannot be accessed. This is also important for preventing malicious data from entering the system by invalidating unsuitable input. |
| 1. Heed Compiler Warnings | By paying attention to compiler warnings, the developer is made aware of issues within the code that may cause the program to behave in unintended manners or not compile at all if the bug is severe enough. Compiler warnings also bring up potential security flaws the developer was not aware of. |
| 1. Architect and Design for Security Policies | Building a system architect allows for distributed code with different levels of privilege. This can help reduce the number of vulnerabilities by not keeping all the sensitive information in just one system, instead distributing it into subsystems. |
| 1. Keep It Simple | Simple code refers to small and non-complex code that is concise and keeps it readable, clean, and reusable. It is easier to debug because it makes things such as unit testing easier than long complex code. |
| 1. Default Deny | The code’s standard should not be to allow access to all and deny invalid inputs. The code should instead deny access to all and only allow valid input. By default, the code should not be looking for what to reject, rather it should be looking for what to accept. |
| 1. Adhere to the Principle of Least Privilege | Processes should be able to be completed with only the bare minimum privileges. Processes need only the privileges that are required for completing the tasks and are set to a strict “need to know” basis. |
| 1. Sanitize Data Sent to Other Systems | All data passed downstream and to subsystems must be sanitized beforehand. Ensure unused functionality does not contain any malicious data such as injections by removing it before it’s send to other system components. |
| 1. Practice Defense in Depth | Reduce security risks by layering defense strategies. Add defenses against vulnerabilities in the physical, technical, and administrative controls. With multiple layers, even if one layer fails, the next layer can protect. |
| 1. Use Effective Quality Assurance Techniques | Practice quality assurance through testing and audits. The quality assurance can be performed internally or by external personnel. Internal reviews usually involve high expertise in the program while external reviews bring a different unbiased perspective. |
| 1. Adopt a Secure Coding Standard | Develop coding standards and security principles based on the programming language and platform being used. Alternatively, apply already established coding standards. Or use a mix of both self-developed and established standards. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Do not assume a positive remainder when using the modulo operator. |

| **Noncompliant Code** |
| --- |
| The insert() function adds values to a buffer in a module fashion, by inserting values at the beginning of the buffer once the end is reached. However, size and index are int therefore not guaranteed to be positive. |
| **int** insert(**int** index, **int** \*list, **int** size, **int** value) {  **if** (size != 0) {      index = (index + 1) % size;      list[index] = value;  **return** index;    }  **else** {  **return** -1;    }  } |

| **Compliant Code** |
| --- |
| The most appropriate solution is to use unsigned types. |
| **int** insert(**size\_t**\* result, **size\_t** index, **int** \*list, **size\_t** size, **int** value) {  **if** (size != 0 && size != SIZE\_MAX) {      index = (index + 1) % size;      list[index] = value;      \*result = index;  **return** 1;    }  **else** {  **return** 0;    }  } |

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

| **Principles(s):**   * **Use effective quality assurance techniques**: With quality testing, simple arithmetic errors like this one can be caught and fixed * **Keep it simple**: simple code makes it more readable and mistakes like this can be caught while reviewing the code |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not modify constant objects |

| **Noncompliant Code** |
| --- |
| The example allows a constant object to be modified. |
| **const** **int** \*\*ipp;  **int** \*ip;  **const** **int** i = 42;    **void** func(**void**) {    ipp = &ip; /\* Constraint violation \*/    \*ipp = &i; /\* Valid \*/    \*ip = 0;   /\* Modifies constant i (was 42) \*/  } |

| **Compliant Code** |
| --- |
| The solution depends on the intent of the programmer. If the intent is that the value of i is modifiable, then it should not be declared as a constant. If the value of i is not meant to change, do not write code that attempts to modify it. |
| **int** \*\*ipp;  **int** \*ip;  **int** i = 42;    **void** func(**void**) {    ipp = &ip; /\* Valid \*/    \*ipp = &i; /\* Valid \*/    \*ip = 0; /\* Valid \*/  } |

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

| **Principles(s):**   * **Heed compiler warnings**: Do not ignore warnings that alert of a possible attempt to modify a const variable |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Range check element access |

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

| **Compliant Code** |
| --- |
| The solution checks that the value returned by get\_index() is within a valid range before calling operator[](). |
| #include <string>    **extern** std::**size\_t** get\_index();    **void** f() {    std::string s("01234567");    std::**size\_t** i = get\_index();  **if** (i < s.length()) {      s[i] = '1';    } **else** {      // Handle error    }  } |

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

| **Principles(s):**   * **Heed compiler warnings**: do not ignore compiler warnings stating an out of range value is being accessed. * **Adopt a secure coding standard**: most secure coding standards should include an in-range check for any value being accessed. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not pass invalid data to the asctime() function. |

| **Noncompliant Code** |
| --- |
| The example invokes the asctime() function with potentially unsanitized data. Unsanitized data can contain malicious code such as SQL injections. |
| #include <time.h>    **void** func(**struct** **tm** \*time\_tm) {  **char** \***time** = **asctime**(time\_tm);    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The strftime() function allows the programmer to specify a more rigorous format and also to specify the maximum size of the resulting time string. It ensures that no more than maxsize characters are printed, preventing buffer overflow. |
| #include <time.h>    **enum** { maxsize = 26 };    **void** func(**struct** **tm** \***time**) {  **char** s[maxsize];    /\* Current time representation for locale \*/  **const** **char** \*format = "%c";    **size\_t** size = **strftime**(s, maxsize, format, **time**);  } |

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

| **Principles(s):**   * **Validate input data**: ensure there is no invalid data in input that may include malicious data such as injections * **Sanitize data sent to other systems**: sanitize information down the pipeline to ensure only valid information is passed down |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| s is dereferenced after it has been deallocated. The vulnerability can be exploited and run arbitrary code with the permissions of the vulnerable process. Typically, dynamic memory allocations and deallocations are far removed, making it difficult to recognize and diagnose such problems. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...  **delete** s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| The dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    **struct** S {  **void** f();  };    **void** g() noexcept(**false**) {    S \*s = **new** S;    // ...    s->f();  **delete** s;  } |

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

| **Principles(s):**.   * **Keep it single**: simple code is easy to follow and keep track of what memory has been deallocated * **Heed compiler warnings**: pay attention to any warnings regarding freed memory access attempts |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| assert() macro is used 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**: properly implement unit testing and regression testing |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not abruptly terminate the program. |

| **Noncompliant Code** |
| --- |
| The call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func() may throw an exception. |
| #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

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

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

| **Principles(s):**   * **Adopt a secure coding standard**: secure coding standards help properly handle exceptions without terminating the program * **Keep it simple**: simple code allows for traceability and proper exception handling |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-008-CPP] | Do not subtract or compare two pointers that do not refer to the same array. |

| **Noncompliant Code** |
| --- |
| In the example, pointer subtraction is used to determine how many free elements are left in the nums array. It incorrectly assumed the nums array is adjacent to the end variable in memory. |
| #include <stddef.h>    **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** end;  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;      /\* Increment next\_num\_ptr as array fills \*/      free\_elements = &end - next\_num\_ptr;  } |

| **Compliant Code** |
| --- |
| In the solution, the number of free elements is computed by subtracting next\_num\_ptr from the address of the pointer past the nums array. While this pointer may not be dereferenced, it may be used in pointer arithmetic. |
| #include <stddef.h>  **enum** { SIZE = 32 };    **void** func(**void**) {  **int** nums[SIZE];  **int** \*next\_num\_ptr = nums;  **size\_t** free\_elements;      /\* Increment next\_num\_ptr as array fills \*/      free\_elements = &(nums[SIZE]) - next\_num\_ptr;  } |

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

| **Principles(s):**   * **Heed compiler warnings**: pay attention to warnings of improper syntax and invalid calls * **Use effective quality assurance techniques**: with proper testing, invalid actions such as arithmetic with improper pointer can be discovered and fixed |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-009-CPP] | Allocate sufficient memory for an object. |

| **Noncompliant Code** |
| --- |
| In the example, inadequate space is allocated for a struct tm object because the size of the pointer is being used to determine the size of the pointed-to object. |
| #include <stdlib.h>  #include <time.h>    **struct** **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {  **struct** **tm** \*tmb;    tmb = (**struct** **tm** \*)**malloc**(**sizeof**(tmb));  **if** (tmb == NULL) {  **return** NULL;    }    \*tmb = (**struct** **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };  **return** tmb; |

| **Compliant Code** |
| --- |
| When allocating space for a single object, passing the dereferenced pointer type to the sizeof operator is a simple way to allocate sufficient memory. Because the sizeof operator does not evaluate its operand, dereferencing an uninitialized or null pointer in this context is well-defined behavior. |
| #include <stdlib.h>  #include <time.h>    **struct** **tm** \*make\_tm(**int** year, **int** mon, **int** day, **int** hour,  **int** min, **int** sec) {  **struct** **tm** \*tmb;    tmb = (**struct** **tm** \*)**malloc**(**sizeof**(\*tmb));  **if** (tmb == NULL) {  **return** NULL;    }    \*tmb = (**struct** **tm**) {      .tm\_sec = sec, .tm\_min = min, .tm\_hour = hour,      .tm\_mday = day, .tm\_mon = mon, .tm\_year = year    };  **return** tmb;  } |

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

| **Principles(s):**   * **Keep it simple**: simple code is easier to follow and easier to track available memory/memory allocation * **Heed compiler warnings**: compiler warnings should alert the programmer of insufficient memory |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-010-CPP] | Detect and remove unused values. |

| **Noncompliant Code** |
| --- |
| In the example, p2 is assigned to the value returned by bar() but that value is never used. |
| **int** \*p1;  **int** \*p2;  p1 = foo();  p2 = bar();    **if** (baz()) {  **return** p1;  }  **else** {    p2 = p1;  }  **return** p2; |

| **Compliant Code** |
| --- |
| The example can be corrected depending on the intent of the programmer. In this solution, p2 is found to be extraneous. The calls to bar() and baz() can be removed if they do not produce any side effects. |
| **int** \*p1 = foo();    /\* Removable if bar() does not produce any side effects \*/  (**void**)bar();    /\* Removable if baz() does not produce any side effects \*/  (**void**)baz();  **return** p1; |

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

| **Principles(s):**   * Sanitize data sent to other systems: |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 | [Insert text.] |
| Encryption at flight | [Insert text.] |
| Encryption in use | [Insert text.] |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | [Insert text.] |
| Authorization | [Insert text.] |
| Accounting | [Insert text.] |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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