**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 data from users is important since you must always consider how bad actors could send something malicious to input fields to try and access the rest of the code. Ways they could do this would be by SQL injections or buffer overloads that fail the systems and could end up producing unpredictable behavior or even output secure information of users. |
| 1. Heed Compiler Warnings | Compiler warnings should be heeded, since they provide useful output to the state of the code. Even if there are technically no compiler errors and the program runs fine, warnings could provide other information such as a method within the code being outdated or unsecure. |
| 1. Architect and Design for Security Policies | Design and architecture for security policies includes designing a website or application’s design and features for secure actions to be structured and only accessible to those who should be allowed to use those features. Features can be secured through user authentication with different user authority levels. |
| 1. Keep It Simple | Keeping code clean and simple is important for people reading the code and also for those who wrote it. This can be done by keeping variable names straightforward to what the variable is supposed to represent. Also condensing long actions into their own functions can improve the readability of the code along with comments from the authors. |
| 1. Default Deny | Default denying actions in code means unless an action is specifically allowed, it is denied. An example of this would be whitelisting characters in a SQL query. Instead of blacklisting certain characters, a set of OK characters are whitelisted and allowed. This way, there can’t be any strange characters we would not have thought to blacklist getting through the prevention function. |
| 1. Adhere to the Principle of Least Privilege | The principle of least privilege means a user whose purpose is to do something like edit data will not be allowed to do anything else with it or outside of that to the larger application. Higher privilege accounts can exist behind secure authentication of those users, but the base app operates with least privileges. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing is the first part of data validation, like the validation of user input mentioned above. Input sanitization is the act of remove anything unsafe from input, like whitelisting characters I mentioned with input validation, and removing and un-whitelisted characters. |
| 1. Practice Defense in Depth | Defense in depth is the practice of adding multiple different defense layers to a program. If one defense layer is breached, the app may not be fully compromised, since theoretically different layers would be there to protect the application. Implementing too many layers could be costly depending on the situation. The amount needed for each app varies widely based on each app’s needs. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques in code include methods such as code review for new features and a test-driven development, including integration testing. Code review could include different teams looking at the code for a new feature; new eyes could reveal some valuable input on the code that may not have been considered. Testing during developing such as unit and integration tests ensure everything is operational during the SDLC. Continuing on the program without testing could end up in a bug being developed early on and not found out about until too late. |
| 1. Adopt a Secure Coding Standard | Adopting a secure coding standard is important to the security of an app since it ensures safe coding practices. Coding standards include information on many different techniques within many different coding languages and how to write them to prevent undefined behavior from the program. |

### 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** | Ensure that unsigned integer operations do not wrap |
| --- | --- | --- |
| **Data Type** | [INT30-C](https://wiki.sei.cmu.edu/confluence/display/c/INT30-C.+Ensure+that+unsigned+integer+operations+do+not+wrap) | Unsigned integers when they over or underflow wrap around. This means if the max of the type is something like 255 and we try to add 1 it will wrap around back to 0 and continue from there if more is added. |

| **Noncompliant Code** |
| --- |
| Code noncompliant with this standard does not check for wrapping with unsigned integers. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {    unsigned int usum = ui\_a + ui\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| To be in compliance with this standard, you must implement an error handler for the occurrence of unsigned integer wrapping. |
| #include <limits.h>    void func(unsigned int ui\_a, unsigned int ui\_b) {    unsigned int usum;    if (UINT\_MAX - ui\_a < ui\_b) {      /\* Handle error \*/    } else {      usum = ui\_a + ui\_b;    }    /\* ... \*/  } |

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

| **Principles(s):** 8 ) Practice defense in depth. Implementing an error handler for unsigned integer wrapping adds a layer of defense against buffer overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | Out of bounds checking / buffer overflow | Cppcheck analyzes code and can detect where buffer overflows could occur. |

#### Coding Standard 2

| **Coding Standard** | **Label** | Ensure that integer conversions do not result in lost or misinterpreted data |
| --- | --- | --- |
| **Data Value** | [INT31-C](https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data) | “Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type.” |

| **Noncompliant Code** |
| --- |
| In this code, converting the unsigned long int to the signed char would result in loss of data. |
| #include <limits.h>    void func(void) {    unsigned long int u\_a = ULONG\_MAX;    signed char sc;    sc = (signed char)u\_a; /\* Cast eliminates warning \*/    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The compliant code checks if the conversion will result in loss of data or sign. |
| #include <limits.h>    void func(void) {    unsigned long int u\_a = ULONG\_MAX;    signed char sc;    if (u\_a <= SCHAR\_MAX) {      sc = (signed char)u\_a;  /\* Cast eliminates warning \*/    } else {      /\* Handle error \*/    }  } |

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

| **Principles(s):** 10 ) adopt a secure coding standard. Type conversion should not be performed when it would result in a type range error |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | Type checks / possible loss of information | Cppcheck is able to spot the possible loss of information from incorrect type conversions |

#### Coding Standard 3

| **Coding Standard** | **Label** | Do not pass a non-null-terminated character sequence to a library function that expects a string |
| --- | --- | --- |
| **String Correctness** | [STR32-C](https://wiki.sei.cmu.edu/confluence/display/c/STR32-C.+Do+not+pass+a+non-null-terminated+character+sequence+to+a+library+function+that+expects+a+string) | If you are going to initialize an array of characters for a string leave the array length empty or remember to account for the null-terminator. |

| **Noncompliant Code** |
| --- |
| Initializing an array of characters to the number of characters within the string you provide will result in a memory error if you forget to account for the null terminator. |
| #include <stdio.h>    void func(void) {    char c\_str[3] = "abc";    printf("%s\n", c\_str);  } |

| **Compliant Code** |
| --- |
| A way to fix this to let the array length be defined by the string we initialize it to so it will recognize the null terminator itself. |
| #include <stdio.h>    void func(void) {    char c\_str[] = "abc";    printf("%s\n", c\_str);  } |

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

| **Principles(s):** 10 ) Adopt a secure coding standard. Initializing an array of characters with the wrong length will results in errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | Bounds checking / check for large enough arrays | If the array length is incorrect, Cppcheck can detect it through its bounds checks |

#### Coding Standard 4

| **Coding Standard** | **Label** | Exclude user input from format strings |
| --- | --- | --- |
| **SQL Injection** | [FIO30-C](https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings) | Do not let un-sanitized user input into a format string |

| **Noncompliant Code** |
| --- |
| In the noncompliant code, untrusted user input is put into a format string argument fprintf(). |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    void incorrect\_password(const char \*user) {    int ret;    /\* User names are restricted to 256 or fewer characters \*/    static const char msg\_format[] = "%s cannot be authenticated.\n";    size\_t len = strlen(user) + sizeof(msg\_format);    char \*msg = (char \*)malloc(len);    if (msg == NULL) {      /\* Handle error \*/    }    ret = snprintf(msg, len, msg\_format, user);    if (ret < 0) {      /\* Handle error \*/    } else if (ret >= len) {      /\* Handle truncated output \*/    }    fprintf(stderr, msg);    free(msg);  } |

| **Compliant Code** |
| --- |
| The compliant version replaces fprintf() with fputs() which outputs the input without evaluating it. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    void incorrect\_password(const char \*user) {    int ret;    /\* User names are restricted to 256 or fewer characters \*/    static const char msg\_format[] = "%s cannot be authenticated.\n";    size\_t len = strlen(user) + sizeof(msg\_format);    char \*msg = (char \*)malloc(len);    if (msg == NULL) {      /\* Handle error \*/    }    ret = snprintf(msg, len, msg\_format, user);    if (ret < 0) {      /\* Handle error \*/    } else if (ret >= len) {      /\* Handle truncated output \*/    }    fputs(msg, stderr);    free(msg);  } |

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

| **Principles(s):** 1 ) Validate input data. User input must be sanitized before it should be in a format string |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| GoogleTest | 1.13.0 | Unit testing | Unit tests on the input can be made to see how the app handles unexpected behavior |

#### Coding Standard 5

| **Coding Standard** | **Label** | Allocate sufficient memory for an object |
| --- | --- | --- |
| **Memory Protection** | [MEM35-C](https://wiki.sei.cmu.edu/confluence/display/c/MEM35-C.+Allocate+sufficient+memory+for+an+object) | Inadequate size arguments for memory allocation could result in a buffer overflow. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code, the struct tm object has inadequate space allocated since the size of the pointer is being used to determine the size of the object pointed to. |
| #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** |
| --- |
| The compliant code fixes the sizing issue by passing the pointer type into malloc(sizeof()) to allocate sufficient memory. |
| #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): 10 )** Adopt a secure coding standard. Proper memory allocation is one of the most important standards for a program to prevent it from crashing |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | Memory leak / bounds checks | Cppcheck has memory leak checks along with the buffer overflow checks mentioned earier |

#### Coding Standard 6

| **Coding Standard** | **Label** | Obey the one-definition rule |
| --- | --- | --- |
| **Assertions** | [DCL60-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule) | Two different declarations, even if they are functionally the same, result in undefined behavior |

| **Noncompliant Code** |
| --- |
| Struct S and class S count as declaring the same thing twice and will result in undefined behavior. |
| // a.cpp  struct S {    int a;  };    // b.cpp  class S {  public:    int a;  }; |

| **Compliant Code** |
| --- |
| Only struct S is used and then it is included in other files, so it is only declared once. |
| // S.h  struct S {    int a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):** 10 ) Adopt a secure coding standard and 4) Keep it simple. The compliant code here is a simpler solution by not having to repeat itself often |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| IDE | Any | Debugging tool / error output | Most IDEs should be able return an error message if the noncompliant code is attempted. |

#### Coding Standard 7

| **Coding Standard** | **Label** | Handle all exceptions |
| --- | --- | --- |
| **Exceptions** | [ERR51-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions) | All exceptions must be caught by the matching exception handler |

| **Noncompliant Code** |
| --- |
| This code terminates if an exception is thrown since there are no exception handlers to throwing\_func(). |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| This is complaint since it tries the function and has a catch method to handle exceptions. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    try {      f();    } catch (...) {      // Handle error    }  } |

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

| **Principles(s):** 8 ) practice defense in depth and 10) adopt a secure coding standard since the code will terminate without an exception handler, and 8 because exception handling is a good layer of defense for DiD. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | Exception safety | Cppcheck detects uncaught exceptions |

#### Coding Standard 8

| **Coding Standard** | **Label** | Write constructor member initializers in the canonical order |
| --- | --- | --- |
| OOP | [OOP53-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP53-CPP.+Write+constructor+member+initializers+in+the+canonical+order) | Constructor member initializers that rely on each other could fail if not ordered correctly |

| **Noncompliant Code** |
| --- |
| “Because the declaration order of the member variables does not match the member initializer order, attempting to read the value of someVal results in an [unspecified value](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-unspecifiedvalue) being stored into dependsOnSomeVal.” |
| class C {    int dependsOnSomeVal;    int someVal;    public:    C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| “This compliant solution changes the declaration order of the class member variables so that the dependency can be ordered properly in the constructor's member initializer list.” |
| class C {    int someVal;    int dependsOnSomeVal;    public:    C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

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

| **Principles(s):** 10 ) Adopt a secure coding standard. Constructor variables that rely on others should be ordered correctly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.10 | Class checks / order of initialization | Cppcheck has checks for classes |

#### Coding Standard 9

| **Coding Standard** | **Label** | Wrap functions that can spuriously wake up in a loop |
| --- | --- | --- |
| Concurrency | [CON54-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CON54-CPP.+Wrap+functions+that+can+spuriously+wake+up+in+a+loop) | “The wait() function must be invoked from a loop that checks whether a [condition predicate](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-conditionpredicate) holds.” |

| **Noncompliant Code** |
| --- |
| “This noncompliant code example nests the call to wait() inside an if block and consequently fails to check the condition predicate after the notification is received. If the notification was spurious or malicious, the thread would wake up prematurely.” |
| #include <condition\_variable>  #include <mutex>    struct Node {    void \*node;    struct Node \*next;  };    static Node list;  static std::mutex m;  static std::condition\_variable condition;    void consume\_list\_element(std::condition\_variable &condition) {    std::unique\_lock<std::mutex> lk(m);      if (list.next == nullptr) {      condition.wait(lk);    }      // Proceed when condition holds.  } |

| **Compliant Code** |
| --- |
| “This compliant solution calls the wait() member function from within a while loop to check the condition both before and after the call to wait().” |
| #include <condition\_variable>  #include <mutex>    struct Node {    void \*node;    struct Node \*next;  };    static Node list;  static std::mutex m;  static std::condition\_variable condition;    void consume\_list\_element() {    std::unique\_lock<std::mutex> lk(m);      while (list.next == nullptr) {      condition.wait(lk);    }      // Proceed when condition holds.  } |

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

| **Principles(s):** 10 ) Adopt a secure coding standard. This is another error-prevention policy |
| --- |

**Threat Level**

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

**Automation**

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

#### Coding Standard 10

| **Coding Standard** | **Label** | Ensure your random number generator is properly seeded |
| --- | --- | --- |
| Random numbers | [MSC51-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC51-CPP.+Ensure+your+random+number+generator+is+properly+seeded) | Using a seed for a random number generator is important because a program using a default seed could end up in the same sequence of numbers being output on multiple runs of the program. |

| **Noncompliant Code** |
| --- |
| This code uses the same default seed for its random number generation. |
| #include <random>  #include <iostream>    void f() {    std::mt19937 engine;      for (int i = 0; i < 10; ++i) {      std::cout << engine() << ", ";    }  } |

| **Compliant Code** |
| --- |
| In the solution, a random value is generate to seed the engine, giving different results each time. |
| #include <random>  #include <iostream>    void f() {    std::random\_device dev;    std::mt19937 engine(dev());      for (int i = 0; i < 10; ++i) {      std::cout << engine() << ", ";    }  } |

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

| **Principles(s):** 10) Adopt a secure coding standard, though this would prioritize really adopting a good testing method for the app to notice if the random number generation was functioning like the programmer wanted it to |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| IDE | Any | Running / debugging | Running the code in a compiler would show the developer the behavior of the random numbers |

### 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.

There are several pieces within this DevSecOps diagram that can become automated. For example, in the verify and test stage at the transition of pre-production and production, things like security testing can be automated. During production in the monitoring and detection phase, automation in DevSecOps has led towards the use of artificial intelligence to help with threat analysis and identify code vulnerabilities in software.

### 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 |
| [FIO30-C](https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings) | High | Likely | Medium | P18 | L1 |
| [INT30-C](https://wiki.sei.cmu.edu/confluence/display/c/INT30-C.+Ensure+that+unsigned+integer+operations+do+not+wrap) | High | Likely | High | P9 | L2 |
| [INT31-C](https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data) | High | Probable | High | P6 | L2 |
| [STR32-C](https://wiki.sei.cmu.edu/confluence/display/c/STR32-C.+Do+not+pass+a+non-null-terminated+character+sequence+to+a+library+function+that+expects+a+string) | High | Probable | Medium | P12 | L1 |
| [MEM35-C](https://wiki.sei.cmu.edu/confluence/display/c/MEM35-C.+Allocate+sufficient+memory+for+an+object) | High | Probable | High | P6 | L2 |
| [MSC51-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC51-CPP.+Ensure+your+random+number+generator+is+properly+seeded) | Medium | Likely | Low | P18 | L1 |
| [ERR51-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions) | Low | Probable | Medium | P4 | L3 |
| [OOP53-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP53-CPP.+Write+constructor+member+initializers+in+the+canonical+order) | Medium | Unlikely | Medium | P4 | L3 |
| [CON54-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CON54-CPP.+Wrap+functions+that+can+spuriously+wake+up+in+a+loop) | Low | Unlikely | Medium | P2 | L3 |
| [DCL60-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule) | 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 | Encryption at rest is the encryption of data stored on a disk. Policies to do this could include encrypting an entire database of an application, select file systems of that database, or select disks, etc. Encryption at rest is important to protect data against those who could get their hands on a company’s physical hardware and try to access it maliciously. |
| Encryption at flight | Encryption at flight protects data in transit. This can be done with Transport Layer Security (TLS) protocols. It should be used since data is vulnerable when it is being transmitted. |
| Encryption in use | In use encryption protects data that is being used or accessed by a user or application. It is the most vulnerable form of data, so it should definitely be prioritizing to encrypt data while it is in use. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | User authentication exists to verify a user accessing an account is who they say they are. Authentication is important since typically accounts can hold sensitive information of their users depending on the site or program’s purpose. Authentication nowadays is typically done with multiple authenticators set up during new user creation or later (2 factor authentication) to have two or more forms of identity checks to verify users. |
| Authorization | Authorization refers to the authorization of privileged users within a network. Common users such as customers on a shopping website should not have access to administrative privileges such as changing database files or ordering stock. This can be secured also through authentication of admin accounts and also distributing privileges in a hierarchy scale from least to most privileges that should be held by different members of staff. |
| Accounting | Accounting keeps track of user tools and activities like user sessions to track resource usage. This helps for security to monitor the network, meaning if there is an anomaly, proper accounting will be able to detect it. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 04/05/2023 | Filled out template | Mitch Sfakianos |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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