**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 | Always, always, always check your input when a user is able to input data into a program. This helps to prevent things like buffer overflows which can lead to an unwanted privilege escalation in the program. This is done by ensuring that the data put in matches what the application is looking for and if not, it throws an error and possibly terminates the application if needed. |
| 1. Heed Compiler Warnings | Compiler warnings are your friend. They tell you what is wrong with your program. Even if it’s only a warning and not an error, make sure you read it and understand it. Then implement what you need to in order to have that warning not appear anymore. |
| 1. Architect and Design for Security Policies | This means that from the very beginning you want to be thinking of security and security policies. The framework that everything will be built on will be a security first approach and it’s to ensure that from the very beginning it will be hard for hackers to get into your system |
| 1. Keep It Simple | This policy means that the security policies and measures that we put into place need to be simple, easy to understand and easy for implementation. The idea here is that if it’s too complicated or too out of the way for the developer to implement security policies then chances are, they won’t which leads to a vulnerable system. So keep it simple and the chances of a developer implementing those security policies are much, much higher. |
| 1. Default Deny | This means that by default we deny all traffic. We then set up filters to allow traffic in but if the traffic doesn’t meet those standards that are set by the filters then it will automatically deny that traffic |
| 1. Adhere to the Principle of Least Privilege | This is a way to restrict privileges only to what is needed for the job role. Nobody has privileges above what is needed and this is so if someone gets targeted for an attack we can minimize the damage that can be caused since they wouldn’t have something like admin privileges if they don’t need those to perform their day-to-day function |
| 1. Sanitize Data Sent to Other Systems | This principle is for ensuring that all data we send out is not malicious or harmful to whatever system we send it to. This is to prevent a hacker from infecting our system and then using malicious code to be sent to every other system on the network. By data sanitizing the code we can mitigate any damage caused by an infected system |
| 1. Practice Defense in Depth | This is the practice of using multiple different approaches to defense and protecting your code base or application from hackers. This would mean using multiple different devices and standards to ensure that you have all your bases cover. IPS’s, Antivirus’s, buffer overflow protection, validating input using all of these together would help to cover much more ground than using a single one by itself. |
| 1. Use Effective Quality Assurance Techniques | This standard means to ensure that there are methods and measures in place to ensure constant and consistent testing of code that is created throughout the development lifecycle. This will help to focus on ensuring that the standards for secure coding are met and that nothing is slipped by within the testing process. |
| 1. Adopt a Secure Coding Standard | This is probably the most important of all the standards. The others are important but if you don’t adopt a secure coding standard in the beginning then it doesn’t really matter. You want to establish how developers can create and publish their code in the most secure way they can and having a standard for them to follow and use will help to make it an easier time for the developers as well as keep consistency between the constant code being created while ensuring that it is being created with a security first approach. |

### 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] | Follow the rule of one-definition. This means that you don’t have multiple variables being called the same with differing definitions |

| **Noncompliant Code** |
| --- |
| This is noncompliant as it had a variable, int a, being used in two different locations as two different things. One is for a struct and the other is for a class. |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  };w |

| **Compliant Code** |
| --- |
| This code is complaint because it creates one definition for the variable, int a is within struct S, within a header file and then any cpp files are able to include that header file and will be able to use that struct S with the variable anywhere in their code. |
| // 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):** Adopt a Secure Coding Standard – This one fits this principle as this is a way to ensure that there is consistency within the coding process as well as it helps to create a secure standard due to not ensuring that each variable has a single name that is not repeated elsewhere for another variable which would create confusion when coding. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++ - DCL60 |  |
| Helix QAC | 2025.1 | C++1067, C++1509, C++1510 |  |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Fully Implemented |
| Parasoft C/C++ test | 2024.2 | CERT\_CPP-DCL60-a | The one definition rule shall not be violated |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Ensure that when you use values you properly check to ensure that the values don’t become too large for it’s unsigned value to help minimize any potential issues that can occur from wrap around. |

| **Noncompliant Code** |
| --- |
| This code is considered noncompliant as it doesn’t check at all if wrap around will occur with the unsigned int variables. If there is wrap around then their can be unexpected results in the program so it is always good to check to ensure that doesn’t happen |
| #include iostream  int addition\_function(unsigned int a, unsigned int b)  {  Return a + b;  } |

| **Compliant Code** |
| --- |
| This function is compliant as it checks to see if an overflow has occurred with unsigned integers a and b and returns true or false based on that. |
| #include iostream  #include limits  bool check\_addition\_overflow(unsigned int a, unsigned int b)  {  return b > numeric\_limits<unsigned int>::max() – a;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard – I believe this one fits into here as there should be a standard in place for numbers and based off of what you need to do, what variable to use, whether it’s unsigned int, unsigned long, or it’s signed variable types and the other variable declarations you can use. This will help to ensure that the correct one is used by developers across the entire project. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | Medium | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Integer-overflow | Fully checked |
| Coverity |  | INTEGER-OVERFLOW | Implemented |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially Implemented |
| TrustInSoft Analyzer | 1.38 | Unsigned overflow | Exhaustively verified |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Ensure you use std::string over char when requiring user input. std::string helps with preventing buffer overflows as strings are dynamically allocated in memory |

| **Noncompliant Code** |
| --- |
| This code creates a char variable and initializes it as input with a length of 20. There are no bounds checking when calling std::cin for user input so the user can input beyond 20 characters which will fill the input variable and lead to a buffer overflow as that data will overwrite the next thing in memory. |
| #include iostream  void function()  {  char input[20];  std::cin >> input;  } |

| **Compliant Code** |
| --- |
| This code is compliant as it uses a string declaration for input which means that when the user inputs data into the string, the program will dynamically allocate the necessary memory to ensure that there is no overflow in memory and thus it prevents buffer overflows |
| #include iostream  void function()  {  std::string input;  std::cin >> input;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard – Okay I know I keep using it but I do feel it works well here because you want a standard when taking in user input to ensure that the user is unable to input too much data that causes a buffer overflow to occur. This coding standard would help to make sure that your application is secure across the board. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | High | 7 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR31 | Detects calls to unsafe string function that may cause buffer overflow |
| Klocwork | 2024.4 | SV.FMT\_STR.BAD\_SCAN\_FORMAT  SV.UNBOUND\_STRING\_INPUT.FUNC |  |
| PC-lint Plus | 1.4 | 421,498 | Partially Supported |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Exhaustively verified |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Ensure that when using any SQL queries that it is validated to ensure that no malicious SQL injections are taking place. Don’t allow any SQL queries to pass by without being properly validated. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant as it performs no checks at all on the sql code and instead would just assume that the sql query is good. You would need to have a check in there before performing the dump\_results function |
| #include <iostream>  void run\_query(std::string& sqlQuery)  {  dump\_results(sqlQuery, dbRecords)  } |

| **Compliant Code** |
| --- |
| This code is compliant as it uses a regex expression to validate the sqlQuery string before returning a boolean on whether or not this string is a malicious SQL Injection or not. It checks for keywords like “and” or “or” which are used in SQL injections to compare things like “WHERE USERNAME IS ROGER OR 1 = 1”. This would come out as true and allow full access to the database if not checked. We check using an if else statement with the return true in the else statement so that the check can’t be bypassed somehow by a malicious hacker |
| #include <iostream>  #include <regex>  bool check\_query(std::string& sqlQuery)  {  std::regex injectCheck("[^0-9A-Z+$][\\s\*(and|or)\\s\*[\\'\"]{0,1}](file:///\\s*(and|or)\\s*%5b\\'\%22%5d%7b0,1%7d)");    if (std::regex\_search(sqlQuery, injectCheck))  {  std::cout << “SQL injection detected.” << std::endl;  return false;  }  else  {  return true;  }  } |

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

| **Principles(s):** Validate Input Data – This one maps to this standard as you want to ensure that all SQL queries are validated to prevent SQL injections. This can be done with using a regex expression to ensure that things like AND and OR are not allowing within the query. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 8 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  | Supported via stubbing |
| LDRA tool suite | 9.7.1 | 86 D | Partially Implemented |
| PC-lint Plus | 1.4 | 592 | Partially Supported |
| Splint | 3.1.1 |  |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Ensure that you are freeing dynamically allocated memory when it is no longer needed in the application |

| **Noncompliant Code** |
| --- |
| This would be an example of noncompliant code as it creates a buffer size of 32 bytes which it then has text\_buffer point to but then it ends the function by returning 0 without freeing any of the dynamically allocated memory in the application. This can result in a memory leak within the program as this memory is “in use” without really being in use. |
| enum { BUFFER\_SIZE = 32 };  int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  }  return 0;  } |

| **Compliant Code** |
| --- |
| This code is considered compliant as it creates a buffer size of 32 bytes and has text\_buffer point to that spot in memory. We then free the memory by using the free method and passing in text\_buffer as the argument |
| enum { BUFFER\_SIZE = 32 };  int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  }    free(text\_buffer);  return 0;  } |

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

| **Principles(s):** Architect and Design for Security Policies – I believe that this maps to this section as I feel that from the very beginning you need to be able to architect how you want memory to be used and how to ensure that all memory not needed is deallocated within the program. This is something that is thought of and designed from the beginning of the project that then is utilized throughout the entire program |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  | Supported but no explicit checker |
| CodeSonar | 9.0p0 | ALLOC.LEAK | Leak |
| Parasoft C/C++ Test | 2024.2 | CERT\_C-MEM31-a | Ensure resources are freed |
| PVS-Studio | 7.37 | V773 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Only use assertions to check logically impossible states. DO NOT use them as a way to check error handling |

| **Noncompliant Code** |
| --- |
| This is noncompliant code as the value of x changes and so when the assert is ran it will crash due to a failed assertion. |
| void function()  {  int x = 7;  x = 9;  assert(x == 7);  } |

| **Compliant Code** |
| --- |
| This is complaint as it checks before any potential changes have been made to the value of x. We wouldn’t want to use assert to check when the variable could have potentially been changed |
| void function()  {  int x = 7;  assert(x == 7);  x = 9;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – This maps perfectly to this as assertions are used to help with QA testing and checking. Though they can easily be misused so you need to ensure you use assertions properly when checking your code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Low | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ERR33 |  |
| Cppcheck Premium | 24.11.0 | Premium-cert-err33-c |  |
| LDRA tool suite | 9.7.1 | 80 D | Partially implemented |
| RuleChecker | 24.04 | Error-information-unused | Partially checked |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Always make sure that when you throw an exception that you also catch the exception. You don’t want to throw something without catching it |

| **Noncompliant Code** |
| --- |
| This code is considered noncompliant as we throw an error but we don’t catch it and display it to the user at all. This can cause the system to abort the program and create an exception message that gets printed to the console. |
| #include iostream  void function(unsigned int a)  {  try  {  if (a % 2 == 0)  {  throw -1;  }  } |

| **Compliant Code** |
| --- |
| This code is complaint as when it throws the error integer it gets catched and displayed to the user on the screen. |
| #include iostream  void function(unsigned int a)  {  try  {  if (a % 2 == 0)  {  throw -1;  }  }    catch (int e)  {  std::cout << “Exception occurred: Error code “ << e << std::endl;  }  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – I am going to map this one to this principle as it goes again with ensuring you use correct QA techniques like for instance, if an exception is thrown you want to make sure it is caught otherwise that can cause issues and unintended effects with the program. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | Error-information-unused  Error-information-unused-computed | Partially Check |
| PC-lint Plus | 1.4 | 534 | Partially Supported |
| PolySpace Bug Finder | R2024b | CERT C: Rule ERR33-C | Rule partially covered |
| TrustInSoft Analyzer | 1.38 | Pointer arithmetic | Exhaustively verified |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Arrays | [STD-008-CPP] | Ensure that all arrays have their bounds specified as it can lead to errors if they aren’t sized properly for what is needed. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant as it doesn’t specify the array bounds at all. This can cause unwanted issues and unexpected results if the initializer for this array changes. |
| int a[] = {1, 2, 3, 4}; |

| **Compliant Code** |
| --- |
| This code correctly specifies the array bound and thus won’t give issues as it’s size matches with the amount inside of the array |
| int a[4] = {1, 2, 3, 4}; |

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

| **Principles(s):** Architect and Design for Security Policies – This one fits here as this is also one that you want to plan for to ensure that your arrays all meet the standard for security policies and don’t end up causing errors and buffer overflows in the future |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | High | 7 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ARR30 | Can detect out-of-bounds access to array/buffer |
| CppCheck | 2.15 | arrayIndexOutOfBounds, outOfBounds, negativeIndex, arrayIndexThenCheck, possible BufferAccessOutOfBounds |  |
| PC-Lint Plus | 1.4 | 413, 415, 416, 613, 661, 662, 676 | Fully Supported |
| TrustedInSoft Analyzer | 1.38 | Index\_in\_address | Exhaustively verified |

Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Pointers | [STD-009-CPP] | Ensure you use proper coding standards when converting a pointer to an integer or an integer to a pointer |

| **Noncompliant Code** |
| --- |
| This code is considered noncompliant as converting a pointer into an int as is isn’t possible due to pointers being 64 bits and unsigned integers being 32 bits. |
| void function()  {  char \*ptr;  unsigned int number = (unsigned int)ptr;  } |

| **Compliant Code** |
| --- |
| This code is considered compliant because it uintptr\_t is an unsigned integer type that can store a data pointer meaning that it is the correct size for converting from a pointer to integer and back. |
| void function()  {  char \*ptr;  uintptr\_t number = (uintptr\_t)ptr;  } |

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

| **Principles(s):** Adopt a Secure Coding Standard – This one maps to this principle as you want to ensure that there are standards in place when converting a pointer to an integer and back again. Without this it could cause issues when developers do it a different way and cause a security issue within the code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT36 | Fully Implemented |
| Coverity | 2017.07 | PW.POINTER\_CONVERSION\_LOSES\_BITS | Fully implemented |
| LDRA tool suite | 9.7.1 | 439 S, 440 S | Fully implemented |
| PC-lint Plus | 1.4 | 4287 | Partially Supported |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Floating-point | [STD-010-CPP] | Do not use floating-point variables as a loop counter |

| **Noncompliant Code** |
| --- |
| This code is noncompliant as it is unclear whether the loop will loop 9 or 10 times due to using a floating-point variable as a loop counter |
| void func()  {  for (float x = 0.1f; x <= 1.0f; x += 0.1f) {  /\* insert code here \*/  }  } |

| **Compliant Code** |
| --- |
| This code is compliant as size\_t is an unsigned integer type and not a floating-point variable. This means that the loop will execute exactly 10 times |
| void func()  {  for (size\_t count = 1; count <= 10; ++count) {  float x = count / 10.0f;  }  } |

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

| **Principles(s):** Adopt a Secure Coding Standard – This one maps to this as you just want to ensure you don’t use floating-point variables as a loop counter as this can cause issues with the loop and it might end early or go for a loop too long causing errors and unintended effects. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Low | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 | For-loop-float | Fully Checked |
| Clang | 3.9 | Cert-flp30-c | Fully Implemented |
| ÉCLAIR | 1.2 | CC2.FLP30 | Fully implemented |
| PC-lint Plus | 1.4 | 9009 | Fully Supported |

### 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 regarding the above diagram I believe that the standards of the policy should be enforced in the DevSecOps process that occurs around the middle of the process. This is the perfect area to ensure enforcement as it comes around during pre-production and it also comes around during production which helps to ensure that in both areas, it is being utilized and properly enforced. Automation of a lot of the standards set though should be sprinkled throughout the entire process. Monitor and detect section can utilize automation in log gathering and in checking to ensure there are no threats within the network using things like a SIEM or an IDS or IPS. We can also use automation within the verifying and testing process of the diagram as automation will help us to more thoroughly run though our tests as well as it will help in speeding up the process of our testing without compromising on the quality of the tests.

### 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 | High | Medium | 4 |
| STD-002-CPP | High | Likely | Medium | Medium | 4 |
| STD-003-CPP | High | Unlikely | Medium | High | 7 |
| STD-004-CPP | High | Likely | High | High | 8 |
| STD-005-CPP | Medium | Probable | Medium | Low | 2 |
| STD-006-CPP | Low | Probable | Low | Low | 2 |
| STD-007-CPP | Medium | Probable | Medium | Low | 3 |
| STD-008-CPP | High | Probable | High | High | 7 |
| STD-009-CPP | Low | Probable | Medium | Low | 3 |
| STD-010-CPP | Low | Probable | Low | Low | 2 |

### 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 at rest | This is where data is stored somewhere like a hard drive and is not currently being used or in transit to somewhere else. The best encryption here is AES-256. You also want to ensure you are using something like Bitlocker or File Vault (on Mac) as if someone breaks into the laptop they tend to go after the data that is at rest on the laptop |
| Encryption in flight | This is when data is on the move through the network. You would want to ensure that the data here is using Transport Layer Security to move through the network. This can mean using protocols like HTTPS or SFTP when transferring data to another location. |
| Encryption in use | This is for data that is currently being used by the device. This you would want to ensure that the data is encrypted using AES-256 as well as it considered one of the best encryption algorithms out there. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is about getting users into the network and how we go about doing it. We can use something like Azure Active Directory or Global Active Directory to be able to create users in the system and check against it when people are trying to get into the network. We can also utilize 802.1X for our network as it requires credentials and certificates to get into the network so if someone isn’t meant to be on it then it won’t allow then access into our systems |
| Authorization | Authorization is about what access users are able to have after authenticating to the network. This means using organizational units to group users and roles into permissions that they are able to have as well as file level permissions per user based on who gave the permissions to who. |
| Accounting | Accounting is pretty much the same as logging. It should be tracking when users login to systems, when there are changes to databases, what files are being accesses, what users are going to on the web and ensure that there is a digital footprint for everything done by every user in the organization. |

**\***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 |