**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 | Ensure that input data is appropriately validated and assigned to a respective type. Never assume that a user will provide correct input. Examples of validating input data can include but are not limited to checking for empty values, checking for correct data types, checking for constraints on input length. |
| 1. Heed Compiler Warnings | A compiler can provide beneficial information to you, letting you know of potential unexpected outcomes before the application is executed. Even though warnings may not prevent an applications execution, you should be aware of and take note of compiler warnings to catch unintended behavior. |
| 1. Architect and Design for Security Policies | Being aware of security policies at the earliest level in an applications life cycle will increase the chances that security is appropriately implemented. When designing/architecting a new solution, or restructuring an existing application, take all security policies into consideration. |
| 1. Keep It Simple | Security vulnerability can be prevented by keeping the complexity of an algorithm low. The greater the complexity, the higher chance that someone may not fully understand what the function is doing. |
| 1. Default Deny | If an object is not explicitly granted access to another object, then the default behavior should be to deny access to the requested object. |
| 1. Adhere to the Principle of Least Privilege | Along the lines of 5. Default Deny, an object should only be granted access at the lowest level it absolutely requires. Try to avoid granting unnecessary access and privileges to avoid chances of an object incorrectly handling data at a high layer when it has no reason to do so. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data sent to other systems ensures that the data will conform to the expectations of downstream systems. It also allows an opportunity to escape any unwanted characters that may alter an applications behavior (ex: SQL Injections). |
| 1. Practice Defense in Depth | Defense in Depth allows multiple layers of security to protect against attacks. If one layer fails, then subsequent layers will attempt to prevent the security vulnerability from succeeding. Be sure not to overestimate or underestimate the layers of security in place. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance techniques will ensure that an application is performing at the highest standards. Quality Assurance can be used to detect unforeseen bugs within an application and can mimic what an end user would attempt. Resolving these bugs found by QA practices leads to a well-rounded and secure application. |
| 1. Adopt a Secure Coding Standard | It is important to be aware of security policies and concerns at every stage of an applications life cycle. Implement security policies and practices into a set of standards that you adhere to and ensure that other members of the team also adhere to these 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 | Use appropriate data types for the functions performed. |

| **Noncompliant Code** |
| --- |
| Ensure that the data type being assigned has appropriate memory allocation for function it is intended to perform.  An unsigned char has a range of 0 to 255, therefore -1 is an invalid value for unsigned char. |
| unsigned char example\_data = -1; |

| **Compliant Code** |
| --- |
| Using the same data type as the previous example, below is a declaration of an unsigned char with a valid value range. |
| unsigned char example\_data = 1; |

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

|  |
| --- |
| **Principles(s):** 1) Validate Input Data – Ensure that appropriate data is assigned to variables and validated before performing logic using them. 2) Keep It Simple – No need to be unnecessarily complex with simple data types will perform the same functions. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ Test | 2021.1 | CERT\_CPP-EXP58-a | Use macros for variable arguments correctly. |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Validate incoming data to ensure that it meets specifications and will not crash an application. |

| **Noncompliant Code** |
| --- |
| The below example will assume that the user input is correct and will assign the value regardless. |
| int age;  cout << “Enter age:”;  cin >> age;  return 0; |

| **Compliant Code** |
| --- |
| Using the isdigit function, the below example will validate that the user input is appropriately an integer before ending the function. |
| int age;  cout << “Enter age:”;  cin >> age;  if (isdigit(age))  {  return 0;  }  return 1; |

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

|  |
| --- |
| **Principles(s):** 1) Validate Input Data – Perform proper validation to verify that the data is in the correct state. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ Test | 2021.1 | CERT\_CPP-EXP58-a | Use macros for variable arguments correctly. |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| The below code will convert the user input into the integer variables i and j. In the event the user input cannot be converted, an error will be thrown. |
| void f() {  int i, j;  std::cin >> i >> j;  } |

| **Compliant Code** |
| --- |
| The above code has been modified to handle any conversion errors when attempting to assign the user input into the integer variables speficied. |
| void f() {  int i, j;    std::cin.exceptions(std::istream::failbit | std::istream::badbit);  try {  std::cin >> i >> j;  // ...  } catch (std::istream::failure &E) {  // Handle error  }  } |

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

|  |
| --- |
| **Principles(s):** 7) Sanitize Data Sent to other systems – Verify that the data is properly handled and used correctly. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR62q |  |
| Clang | 3.9 | Cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C standard library functions corresponding to ERR34-C |
| CodeSonar | 6.1p0 | BADFUNC.ATOF | Use of atof |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-ERR62-a | The library functions atof, atoi and atoll from library stdlib.h shall not be used |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Ensure the queries are appropriately structured to prevent SQL injections. |

| **Noncompliant Code** |
| --- |
| The below code contains a SQL query that will return all column results where the first name equals ‘Marco’, with the caveat being that 1 = 1 is appended to the end. Because of this, all records will be returned because the ‘or’ operator will return results where either statement is true. |
| String query = “SELECT \* FROM Users WHERE first\_name = ‘Marco’ or 1=1’”; |

| **Compliant Code** |
| --- |
| By switching to a parameterized format, the below code will now only execute with the value assigned to first\_name; |
| String first\_name, query;  cout << “Enter first name”;  cin >> first\_name;  if (first\_name == “Marco”)  {  Query = “SELECT \* FROM Users WHERE first\_name = “+ first\_name;  } |

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

|  |
| --- |
| **Principles(s):** 1) Validate Input Data – Validate the incoming parameters for the SQL query to verify that no potential SQL injections will occur. 3) Architect and Design for Security Policies – When designing an application that will utilize SQL queries, take precaution to limit the possibilities for potential SQL injections. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| CodeSonar | 6.1p0 | JAVA.IO.INJ.SQL | SQL Injection (Java) |
| Coverity | 7.5 | SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED | Implemented |
| SonarQube | 6.7 | S2077 S3649 | Executing SQL queries is security-sensitive  SQL Queries should not be vulnerable to injection attacks |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Adequately ensure that memory allocation is considered when planning variable/array declarations to avoid memory leaks (ie: Buffer Overflows) |

| **Noncompliant Code** |
| --- |
| The below code accepts a user input and assigns it to a variable without checking for length. In the event that the input exceeds 20 characters, a buffer overflow will occur. |
| char user\_input[20];  cout << “Enter a value: “;  cin >> user\_input; |

| **Compliant Code** |
| --- |
| The above code has been modified to use a function which limits the number of characters to an amount specified. The remainder of the input that exceeds this character limit will be ignored. |
| char user\_input[20];  cout << “Enter a value: “;  cin.getline(user\_input, 20); |

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

|  |
| --- |
| **Principles(s):** 2) Heed Compiler Warnings – Typically, compilers will alert you for potential memory allocation issues. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2021.1 | 433, 826 | Partially Supported |
| PRQA QA-C++ | 4.4 | 2840, 2841, 2842, 2843, 2935, 2936, 2937, 2938 |  |
| RuleChecker | 20.10 | Malloc-size-insufficient | Partially checked |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Exhaustively detects undefined behavior |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Use assertions to verify that code is performing as expected. |

| **Noncompliant Code** |
| --- |
| The below code will attempt to print a number, without first asserting if it has a value. |
| void output(int\* myInt)  {  cout << myInt;  } |

| **Compliant Code** |
| --- |
| The above code has been modified to include an assertion that the myInt variable is not null. This acts as a debugging tool to verify code behaves as expected and can let developers know at what point a program is failing. |
| #include <assert.h> // Assert header file  void output(int\* myInt)  {  assert(myInt != NULL);  cout << myInt;  } |

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

|  |
| --- |
| **Principles(s):** Use Effective Quality Assurance Techniques – Assertions can be used as an easy method to verify code is behaving as an application runs. Typically, Assertions are removed on a release build. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully Implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully Implemented |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Make sure that proper exception handling is in place to prevent an unexpected application crash. |

| **Noncompliant Code** |
| --- |
| The below example will not catch the exceptions thrown by the test method exception\_example. |
| void exception\_example() noexcept(false);  void testMethod()  {  exception\_example(); // Will throw exception  }  int main()  {  testMethod(); // No error handling in place.  } |

| **Compliant Code** |
| --- |
| The above example has been modified to include a try/catch block to handle the exception being thrown in the main method, which will prevent an unexpected application crash. |
| void exception\_example() noexcept(false);  void testMethod()  {  exception\_example(); // Will throw exception  }  int main()  {  try {  testMethod();  }  catch (…){  // Handle exception.  }  } |

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

|  |
| --- |
| **Principles(s):** 2) Heed Compiler Warnings – Typically, compilers will warn about potential exceptions being thrown. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-ERR56-a | Ensure resources are freed |
| PRQA QA-C++ | 4.4 | 4075, 4076 |  |
| PVS-Studio | 7.13 | V565, V1023, V5002 |  |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Cyclomatic Complexity** | STD-008-CPP | Ensure that methods or functions are maintainable by reducing the number of potential outcomes and keeping the cyclomatic complexity below 15. |

| **Noncompliant Code** |
| --- |
| The below code contains nested if statements that can be difficult to read and hard to maintain. |
| void testFunction(int\* myInt)  {  If (myInt != NULL)  {  If (myInt < 5)  {  // Do logic (potentially more nesting)  }  }  } |

| **Compliant Code** |
| --- |
| The above code has been modified to return early and limit the amount of nesting within a function. This allows a given function or method to be easier to read and maintain. |
| void testFunction(int\* myInt)  {  If (myInt == NULL)  {  return;  }  If (myInt < 5)  {  // Do logic.  }  } |

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

|  |
| --- |
| **Principles(s):** 4) Keep it simple – Cyclomatic complexity is usually a result of overly complex systems. Try to architect the application to avoid complexity when applicable. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| ÉCLAIR | 1.2 | CC2.EXP30 | Fully Implemented |
| SonarQube C/C++ Plugin | 4.10 | IncAndDecMixedWithOtherOperators | Partially Implemented |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Declaration and Initializations** | STD-009-CPP | Do not modify the standard namespaces. |

| **Noncompliant Code** |
| --- |
| The below code will declare a variable within the std namespace which will result in undefined behavior. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| The above code has been modified to declare the variable x in a namespace that is **not** the std namespace. |
| namespace nonStd {  int x;  } |

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

|  |
| --- |
| **Principles(s):** 10) Adopt a secure coding standard – Namespaces such as ‘std’ are provided for us and therefore should not be altered as a standard. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL58 |  |
| Parasoft C/C++test | 2021.1 | CERT.DCL.STD\_NS\_MODIFIED | Do not modify the standard namespace ‘std’ and ‘posix’ |
| Polyspace Bug Finder | R2021a | CERT C++:CDL58-CPP | Checks for modification of standard namespaces (rule fully covered) |
| SonarQube C/C++ plugin | 4.10 | S3470 |  |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Object Oriented Programming** | STD-010-CPP | Write constructor member initializers in the canonical order. |

| **Noncompliant Code** |
| --- |
| The below code contains a constructor that attempts to initialize a function someVa() first, then to initialize dependsOnSomeVal() to a value dependent by someVal(). Because the order does not match the initializer order, attempting to read the value from someVal() will result in an unspecified value. |
| class C {  int dependsOnSomeVal;  int someVal;    public:  C(int val) : someVal(val), dependsOnSomeVal(someVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| The above code has been modified to correct the declaration order of the class member variables so that dependsOnSomeVal() can correctly rely on someVal() |
| 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):** Architect and Design for Security Policies – Structure of classes in Object Oriented Program should be well thought out and designed to ensure they meet the necessary requirements. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Initializer-list-order | Fully Checked |
| LDRA tool Suite | 9.7.1 | 206 S | Fully Implemented |
| Parasoft C/C++ | 2021.1 | CERT\_CPP-OOP53-a | Checks for members not initialized in canonical order (rule fully covered) |
| RuleChecker | 20.10 | Initializer-list-order | Fully Checked |

## Defense-in-Depth Illustration

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



# Project One

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

## Revise the C/C++ Standards

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

## Risk Assessment

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

## Automated Detection

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

## Automation

Provide a written explanation using the image provided.



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

* Many of the security principles defined in this policy can be applied to the Pre-Production phase of Green Pace’s current DevSecOps infrastructure. The policies defined here can be implemented in the Design and Build phase of Pre-Production as it revolves heavily around being aware and conscious of secure coding principles. It is important for Green Pace to be thoughtful and implement the security policies within this document as early as possible, and to continue keeping security as a focus on the entire lifespan of the application.

## 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 | Medium | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | Medium | Likely | Medium | P4 | L3 |
| STD-003-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-004-CPP | High | Probable | Medium | P12 | L1 |
| STD-005-CPP | High | Probable | High | P6 | L2 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | High | Likely | High | P9 | L2 |
| STD-008-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-009-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |

## Create Policies for Encryption and Triple A

Include all three types of encryptions (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 designed to prevent the attacker from accessing the unencrypted data by ensuring the data is encrypted when on the disk. Data such as passwords or other personal information should remain encrypted and properly stored when not in use. |
| Encryption at flight | Encryption in flight is designed to encrypt the data during data transmission over a network. This is to keep the data secure in the event of a network injection attack. |
| Encryption in use | Like Encryption in Rest, Encryption in Use is designed to ensure that sensitive data is never left unsecured. Either data in use, or data at rest should remain encrypted for the best security practice. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication allows a user to be identified, traditionally though the use of a username/password combination. Typically, authentication is enforced using security challenge questions to verify a user is who they say they are when logging in. |
| Authorization | Authorization grants user the ability to do certain tasks. A user’s authorization can allow them to access files, executing queries or stored procedures on a database, or perform certain functions that require an elevated privilege of access. |
| Accounting | Accounting measures the resources a user consumes during access. This typically consists of logging what a user performs and capturing any anomalies or errors. |

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

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

## Map the Principles

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

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

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

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

# Audit Controls and Management

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

Evidence will include the following:

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

# Enforcement

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

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

# Exceptions Process

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

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

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

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

# Distribution

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

# Policy Change Control

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

# Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 08/08/2021 | Completing Security Policy | Marco Aldana | [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 |

**Resources**

“Assertions.” *DCL03-C. Use a Static Assertion to Test the Value of a Constant Expression - SEI CERT C Coding Standard - Confluence*, wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression.

“CERT C++: Data Validation.” *EXP58-CPP. Pass an Object of the Correct Type to va\_start - SEI CERT C++ Coding Standard - Confluence*, wiki.sei.cmu.edu/confluence/display/cplusplus/EXP58-CPP.+Pass+an+object+of+the+correct+type+to+va\_start.

“CERT C++: String Correctness.” *ERR62-CPP. Detect Errors When Converting a String to a Number - SEI CERT C++ Coding Standard - Confluence*, wiki.sei.cmu.edu/confluence/display/cplusplus/ERR62-CPP.+Detect+errors+when+converting+a+string+to+a+number.

Contributor, TechTarget. “What Is Authentication, Authorization, and Accounting (AAA)? - Definition from Whatis.com.” *SearchSecurity*, TechTarget, 16 Nov. 2010, searchsecurity.techtarget.com/definition/authentication-authorization-and-accounting.

“Cyclomatic Complexity.” *EXP50-CPP. Do Not Depend on the Order of Evaluation for Side Effects - SEI CERT C++ Coding Standard - Confluence*, wiki.sei.cmu.edu/confluence/display/cplusplus/EXP50-CPP.+Do+not+depend+on+the+order+of+evaluation+for+side+effects.

“Data Privacy and Security.” *Quantum Metric*, 9 July 2021, www.quantummetric.com/platform-foundations/data-privacy-security/.

“Exceptions.” *ERR56-CPP. Guarantee Exception Safety - SEI CERT C++ Coding Standard - Confluence*, wiki.sei.cmu.edu/confluence/display/cplusplus/ERR56-CPP.+Guarantee+exception+safety.

“Memory Allocation.” *MEM35-C. Allocate Sufficient Memory for an Object - SEI CERT C Coding Standard - Confluence*, wiki.sei.cmu.edu/confluence/display/c/MEM35-C.+Allocate+sufficient+memory+for+an+object.

Msmbaldwin. “Azure Data Encryption-at-Rest - Azure Security.” *Azure Data Encryption-at-Rest - Azure Security | Microsoft Docs*, docs.microsoft.com/en-us/azure/security/fundamentals/encryption-atrest#:~:text=Encryption%20at%20rest%20is%20designed,encryption%20to%20read%20the%20data.

“Namespaces.” *DCL58-CPP. Do Not Modify the Standard Namespaces - SEI CERT C++ Coding Standard - Confluence*, wiki.sei.cmu.edu/confluence/display/cplusplus/DCL58-CPP.+Do+not+modify+the+standard+namespaces.

“Object-Oriented Programming.” *OOP53-CPP. Write Constructor Member Initializers in the Canonical Order - SEI CERT C++ Coding Standard - Confluence*, wiki.sei.cmu.edu/confluence/display/cplusplus/OOP53-CPP.+Write+constructor+member+initializers+in+the+canonical+order.

“SQL Injection.” *IDS00-J. Prevent SQL Injection - SEI CERT Oracle Coding Standard for Java - Confluence*, wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection.