**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 | Validate that the data received from users is formatted correctly and does not allow the user to manipulate or attack sensitive code such as databases. |
| 1. Heed Compiler Warnings | Do not leave any compiler warnings in the production of the final code, compiler warnings are mistakes that can potentially cause problems or security breaches. |
| 1. Architect and Design for Security Policies | Designing a system to prevent security breaches involves layering security for multiple purposes and isolating breaches through architecture that prevents in the case of breaches less damage to be done. |
| 1. Keep It Simple | More complex code has more vulnerabilities, simple is easier to prevent and diagnosis issues. |
| 1. Default Deny | Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted |
| 1. Adhere to the Principle of Least Privilege | Users should only have access to the services they are required to do their job. This limits the damage to breaches. |
| 1. Sanitize Data Sent to Other Systems | Modify validated user inputs to prevent harmful data content, disallowing dangerous characters or tags that could be used to inject harmful cod.e |
| 1. Practice Defense in Depth | Include multiple layers of security that protect the system in different ways and from different attacks that includes overlap. An example, Firewall, Anti-Virus and Anti-Cheat |
| 1. Use Effective Quality Assurance Techniques | Implement a community of positive peer reviews, code analysis tools and unit testing and update your security profiles regularly to keep security standards up to date. |
| 1. Adopt a Secure Coding Standard | Adopt standards within the company that uses the methods above to prevent security breaches that should be implemented along common sense security internal practices. |

### 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** | **Do not define C-Style Variadic function** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Allowing variables of multiple unknown types can lead to insertion of unintended data types that could create erroneous behavior |

| **Noncompliant Code** |
| --- |
| In the given example if any data type other than a integer is passed into the function a undefined behavior will occur. |
| #include <cstdarg>  int add(int first, int second, ...) {  int r = first + second;  va\_list va;  va\_start(va, second);  while (int v = va\_arg(va, int)) {  r += v;  }  va\_end(va);  return r;  } |

| **Compliant Code** |
| --- |
| The given compliant examples allows for an undefined number of variables but limits them to a specific type if validated in the method. |
| #include <type\_traits>    template <typename Arg, typename... Ts, typename std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  int add(Arg i, Arg j, Ts... all) {  int values[] = { j, all... };  int r = i;  for (auto v : values) {  r += v;  }  return r; |

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

| **Principles(s):** This coding standard is about validating input data it ensures that the code writer is inputting the correct amount of variables. This is a core to preventing a SQL injection one of the most common types of attacks. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Highly | Low | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

#### Coding Standard 2

| **Coding Standard** | **Label** | Do Not Cast an out-of-range enumeration value |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do Not Cast an out-of-rang enumeration value |

| **Noncompliant Code** |
| --- |
| In then non-compliant code the enum is used without applying a range to the value as enum is a undefined number by default. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| This sets the enums to remain within the range of an integer but could be done with similar numerical data types. |
| enum EnumType : int {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** This principle represents number 5 the default deny, if the results are outside the expected inputs we should reject it and have a plan for handling the error. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Medium | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Copying data that is not large enough to hold that data results in a buffer overflow |

| **Noncompliant Code** |
| --- |
| This code allows for the user to input data that is greater than the scope of the character strings. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| This try/catch being applied will check if the characters provided are within the size buffer and then place inside a string. |
| #include <fstream>  #include <string>  void f(std::istream &in) {  char buffer[32];  try {  in.read(buffer, sizeof(buffer));  } catch (std::ios\_base::failure &e) {  // Handle error  }  std::string str(buffer, in.gcount());  // ...  } |

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

| **Principles(s):** This applies to both Validate Data Input and Sanitize Data Sent, by converting the longer data into a string we are hitting two potential problems at once we are preventing invalid (cropped) data entries and also converting data into strings before testing them which helps against sequel injection |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Exclude user input from format strings** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-LLL] | To prevent malicious users from using a tainted value string that could allow unintended access. |

| **Noncompliant Code** |
| --- |
| Using the fprint() function evaluates the string in order to print it thus creating the vulnerability. |
| #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** |
| --- |
| By replacing fprints with fputs |
| #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):** Validate Input Data, SQL injection is the most common type of attack making sure any characters read by the database are not interpreted as a code will remain a top priority. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do not access freed memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | When deallocating data in a linked list be sure to allocate a new location in the list |

| **Noncompliant Code** |
| --- |
| In this example they do not connect the node to the next node before freeing the node breaking the linked list. |
| #include <stdlib.h>    struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  for (struct node \*p = head; p != NULL; p = p->next) {  free(p);  }  } |

| **Compliant Code** |
| --- |
| In this example the connect the previous node to the targeted nodes next node before freeing it. |
| #include <stdlib.h>    struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  struct node \*q;  for (struct node \*p = head; p != NULL; p = q) {  q = p->next;  free(p);  }  } |

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

| **Principles(s):** Use effective quality assurance and keep it simple, this is core linked list practice to prevent loss of data when deletions happen its important to link the data that remains in the database. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Run Assertion to verify your code** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Run assertion in your code base before adding to production. It is important to test that your expected failed results fail as intended. |

| **Noncompliant Code** |
| --- |
| Assert tests should not fail by default. |
| #include <iostream>  // uncomment to disable assert()  // #define NDEBUG  #include <cassert>    // Use (void) to silence unused warnings.  #define assertm(exp, msg) assert((void(msg), exp))    int main()  {  assert((void("void helps to avoid 'unused value' warning"), 2 \* 2 == 4));  std::cout << "Checkpoint #2\n";  } |

| **Compliant Code** |
| --- |
| Make assertion tests that assert both successes and failures. |
| #include <iostream>  // uncomment to disable assert()  // #define NDEBUG  #include <cassert>    // Use (void) to silence unused warnings.  #define assertm(exp, msg) assert((void(msg), exp))    int main()  {  assert(2 + 2 == 4);  [std::cout](http://en.cppreference.com/w/cpp/io/cout) << "Checkpoint #1**\n**";    assert((010 + 010 == 16) && "Yet another way to add an assert message");  [std::cout](http://en.cppreference.com/w/cpp/io/cout) << "Checkpoint #3**\n**";    assertm((2 + 2) % 3 == 1, "Success");  [std::cout](http://en.cppreference.com/w/cpp/io/cout) << "Checkpoint #4**\n**";    assertm(2 + 2 == 5, "Failed"); // assertion fails  [std::cout](http://en.cppreference.com/w/cpp/io/cout) << "Execution continues past the last assert**\n**";  // No output |
|  |

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

| **Principles(s):** 9 Use Effective Quality Assurance – This primarily involves testing the code before launching it so that you are prepared for a variety of edge cases. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Moderate | High | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle All Exeception** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Its important to make sure when an error occurs that it Is caught using try/catch and handles the error. |

| **Noncompliant Code** |
| --- |
| The is is an example of code that doesn’t respond if the function fails. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| Using try/catch to report when the code fails with the appropriate handling |
| 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):** This falls under the Keep it simple principle as well as the architect and design, using try/catch as a method is a priority in handling errors, always handling your errors is a simple logic that can be applied universally. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Moderate | Medium | Medium | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Do not leak resources when handling exceptions** |
| --- | --- | --- |
| Memory/Exception Handling | [STD-008-LLL] | Cleanup the variables after a try/catch error is handled. If an uncompleted job has thrown an exception could cause a leak |

| **Noncompliant Code** |
| --- |
| This is an example of the code that handles the exception but doesn’t clear failed variables. |
| #include <new>    struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  void process\_item() noexcept(false);  };    void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }    try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  throw;  }  delete pst;  } |

| **Compliant Code** |
| --- |
| Handles errors and deletes PST before leaving the section. |
| #include <new>  struct SomeType {  SomeType() noexcept; // Performs nontrivial initialization.  ~SomeType(); // Performs nontrivial finalization.  void process\_item() noexcept(false);  };  void f() {  SomeType \*pst = new (std::nothrow) SomeType();  if (!pst) {  // Handle error  return;  }  try {  pst->process\_item();  } catch (...) {  // Process error, but do not recover from it; rethrow.  delete pst;  throw;  }  delete pst;  } |

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

| **Principles(s):** This focuses on the Sanitize the Data sent principles after a code has been run cleaning up inputs prevents memory leaks which can lead to crashes in the event of an attack intended to stop services. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Moderate | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Properly deallocate dynamically allocated resources** |
| --- | --- | --- |
| Memory Management | [STD-009-CPP] | One memory is no longer being used deallocating memory afterwords will keep the program running smoothly utilizing deconstructors. |

| **Noncompliant Code** |
| --- |
| No deconstruct or is being used so the empty memory slot is still allocated |
| [#include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;  // ...  delete s1;  } |

| **Compliant Code** |
| --- |
| By using deconstructors you are removing the memory slot after the memory is no longer in use. |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...  s1->~S();  } |

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

| **Principles(s):** This primarily focuses on the Sanitize the Data sent principle, not deconstructing data when delated will lead to memory bloats and crashes. This can be utilized in an attack by malicious entities to deny services. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Moderate | Low | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do not store an already-owned pointer value in an unrelated smart pointer** |
| --- | --- | --- |
| Memory Management | [STD-010-LLL] | By creating a multiple smart pointers that reference the same original pointer, when one of the pointers is delated it will delete the data of the second pointer leaving a empty pointer |

| **Noncompliant Code** |
| --- |
| If P1 where to be deleted, P2 reference would also be affected creating an empty p2 |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| If you want to have a duplicate case, make a copy rather than referencing the same data. |
| #include <memory>  void f() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1);  } |

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

| **Principles(s):** This focuses on Keeping it simple and adhere to principle of least privilege in the case of a user profiles, If the pointer your referencing is a user this could lead to adding privileges to a new user if the original pointer was updated to have a new privilege that was unintended. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **C++ static code analyzer** | C++ | Sonar Source | **C++ static code analyzer** |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

### Summary of Risk Assessments

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

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

### 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 | Data is at rest when it is stored this can be in the cloud, or a hard drive that is not being used or viewed by its intended user. This means that any data hard-stored in a company database must be encrypted. This way if information is accessed without a correct token it will be unable to be read. It involves using a secure encrypted database such as mongo DB and never send data that is unencrypted. |
| Encryption in flight | Data that is encrypted between devices accompanied by an Access Token that prevents an illegitimate user from decrypting the data correctly on site.  It practices that involves giving users and unique access token that that will decrypt the information that they request. |
| Encryption in use | Encryption in use of data that has been received and is decrypted for an Authenticated and authorized user.  In practice this involves limited access to very sensitive data higher level employees. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of making the intended subjects are the ones accessing the data are who you want to be accessing them.  We will use multi-factor authentication that includes a username and password and a paired company device to validate the user’s id. |
| Authorization | It is about validating what the intended user is doing what they are intending to do.  Each user will have an access token, that is tied to their account their user access will be used to handle the encryption for what files they transmit. While also blocking their information if it is used as the key to decrypt if an invalid key is used to decrypt their information a false one will be generated. |
| Accounting | Involves keeping records of what occurs within the program. So that when a breech or error occurs, we can find what caused it.  We will do this by keeping records of when a user logins in, makes a change or retrieves data from the database, adds or removes a user, what their level of access is and what files they access. |

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 05/20/2025 | Coding Standard Choices | CJ Cline | Philomena Ogoh |
| 3.0 | 06/14/2025 | Final Revisions | CJ Cline | Philomena Ogoh |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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