

# Green Pace Secure Development Policy

# Hayley Blake

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating input data is crucial in limiting the vulnerabilities that can arise from user input. We do this by validating the syntax and semantics of all inputs. Check syntax to ensure that it is the correct format and that other values that don’t match cannot be input. Check semantics by limiting values that fall within the edge cases. |
| 1. Heed Compiler Warnings | When compiling code, it is necessary to address and take precautions against arising warnings. Use static analysis tools to assess code and find warnings. Lastly, manual code review will be used to test for warnings and assess code stability. |
| 1. Architect and Design for Security Policies | The security policies will start with the architecture and design of programs. Security ground rules for software will begin in the design phase. These standards will be according to accredited frameworks such as NIST and ISO 27001. |
| 1. Keep It Simple | Software should be built in an organized, non-overly complicated way. When creating software, it is necessary to comment code to explain what each part does so that it can be found quickly. |
| 1. Default Deny | When creating default permissions, it is important to take the cautious route and set it to deny. The way this works is that all actions that are not explicitly permitted will be automatically rejected. Users of the software will be denied permission unless they have the right roles. |
| 1. Adhere to the Principle of Least Privilege | The principle of Least Privilege gives users the minimum permissions needed to perform a task. In software, it is necessary to limit access to different parts to reduce the number of vulnerabilities. Giving only the minimum required permissions will protect the system from compromise. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data is important because we don’t want private data being exposed when it is sent to other systems. When data is sent to systems that are allowed to access it, it should be encrypted so that only those systems can read it. Access to these files should always be temporary, and once the limit is up, the data should be erased, and the other system will have to ask for permission again. |
| 1. Practice Defense in Depth | Defense in Depth should be applied to the software in layers. Each layer will address a different vulnerability in the system. This will allow us to have multiple system blocks in case of an attack and to handle them. |
| 1. Use Effective Quality Assurance Techniques | Using Quality Assurance techniques such as penetration testing, risk assessment, secure coding, monitoring, and auditing. Those are just some of the techniques that should be used to test the software at all stages of development. It is especially important to keep testing after updates and to continue testing to ensure we don’t fall behind on cybersecurity. |
| 1. Adopt a Secure Coding Standard | The purpose of adopting secure coding standards is to protect against vulnerabilities, bugs, and other errors. We will be using the CERT, OWASP, and other standards as they evolve. |

### 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-CLG | Range check element access |

| **Noncompliant Code** |
| --- |
| The value being stored is greater than the value stored in the string; this will cause undetermined behavior. |
| #include <string>    extern std::size\_t get\_index();    void f() {  std::string s("01234567");  s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| Create a catch statement to send an error and stop the overflow from happening. |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();    void f() {  std::string s("01234567");  try {  s.at(get\_index()) = '1';  } catch (std::out\_of\_range &) {  // Handle error  }  } |

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

| **Principles(s):** 1, 3, 10 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | assert\_failure |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CLG | Do not use input functions to convert character data if they cannot handle all possible inputs |

| **Noncompliant Code** |
| --- |
| The code scans for input from a string, then it tries to convert it into a long, which causes an error. |
| long num\_long;    if (scanf("%ld", &num\_long) != 1) {  /\* Handle error \*/  } |

| **Compliant Code** |
| --- |
| This first creates a reader and an error for that. Then multiple else if statements to catch the rest of the errors. |
| char buff[25];  char \*end\_ptr;  long num\_long;    if (fgets(buff, sizeof(buff), stdin) == NULL) {  if (puts("EOF or read error\n") == EOF) {  /\* Handle error \*/  }  } else {  errno = 0;    num\_long = strtol(buff, &end\_ptr, 10);    if (ERANGE == errno) {  if (puts("number out of range\n") == EOF) {  /\* Handle error \*/  }  }  else if (end\_ptr == buff) {  if (puts("not valid numeric input\n") == EOF) {  /\* Handle error \*/  }  }  else if ('\n' != \*end\_ptr && '\0' != \*end\_ptr) {  if (puts("extra characters on input line\n") == EOF) {  /\* Handle error \*/  }  }  } |

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

| **Principles(s):** 1, 2 4, 9, 10 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | MISC.NEGCHAR | Negative Character Value |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Use valid references, pointers, and iterators to reference elements of a basic\_string |

| **Noncompliant Code** |
| --- |
| The first insert works, but any after become undefined. |
| #include <string>    void f(const std::string &input) {  std::string email;    // Copy input into email converting ";" to " "  std::string::iterator loc = email.begin();  for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {  email.insert(loc, \*i != ';' ? \*i : ' ');  }  } |

| **Compliant Code** |
| --- |
| We call loc again to assign a value to the string. |
| #include <string>    void f(const std::string &input) {  std::string email;    // Copy input into email converting ";" to " "  std::string::iterator loc = email.begin();  for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {  loc = email.insert(loc, \*i != ';' ? \*i : ' ');  }  } |

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

| **Principles(s):** 4, 7, 10 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | ALLOC.UAF | Use After Free |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-JAV | Prevent SQL Injection |

| **Noncompliant Code** |
| --- |
| Stores the password as a char, making it insecure and creating susceptibility to SQL attacks. |
| class Login {  public Connection getConnection() throws SQLException {  DriverManager.registerDriver(new  com.microsoft.sqlserver.jdbc.SQLServerDriver());  String dbConnection =  PropertyManager.getProperty("db.connection");  // Can hold some value like  // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  return DriverManager.getConnection(dbConnection);  }    String hashPassword(char[] password) {  // Create hash of password  }    public void doPrivilegedAction(  String username, char[] password  ) throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);  String sqlString = "select \* from db\_user where username=" +  username + " and password =" + pwd;  PreparedStatement stmt = connection.prepareStatement(sqlString);    ResultSet rs = stmt.executeQuery();  if (!rs.next()) {  throw new SecurityException("User name or password incorrect");  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  }  } |

| **Compliant Code** |
| --- |
| Uses query with ? instead so that it doesn’t call a SQL attack |
| public void doPrivilegedAction(  String username, char[] password  ) throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);    // Validate username length  if (username.length() > 8) {  // Handle error  }    String sqlString =  "select \* from db\_user where username=? and password=?";  PreparedStatement stmt = connection.prepareStatement(sqlString);  stmt.setString(1, username);  stmt.setString(2, pwd);  ResultSet rs = stmt.executeQuery();  if (!rs.next()) {  throw new SecurityException("User name or password incorrect");  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  } |

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

| **Principles(s):** 1, 3, 4, 5, 7, 8 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.0p0 | JAVA.IO.INJ.SQL | SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| Creates a new entry into the array but doesn’t check array size |
| #include <cstring>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new int[size];  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy;  } |

| **Compliant Code** |
| --- |
| Handles array error if the array cannot be copied again |
| #include <cstring>  #include <new>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = new (std::nothrow) int[size];  if (!copy) {  // Handle error  return;  }  std::memcpy(copy, array, size \* sizeof(\*copy));  // ...  delete [] copy;  } |

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

| **Principles(s):** 1, 8, 9 |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P27 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CLG | Incorporate diagnostic tests using assertions |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;    len = strlen(c\_str);  dup = (char \*)malloc(len + 1);  assert(NULL != dup);    memcpy(dup, c\_str, len + 1);  return dup;  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;    len = strlen(c\_str);  dup = (char\*)malloc(len + 1);  /\* Detect and handle memory allocation error \*/  if (NULL == dup) {  return NULL;  }    memcpy(dup, c\_str, len + 1);  return dup;  } |

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

| **Principles(s):** 3, 5, 8, 9 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | LANG.FUNCS.ASSERTS | Not enough assertions |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| Just hard-codes the throw without proper catching mechanisms |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In the main, a try/catch happens so that the error is shown and caught. |
| 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):** 1, 3, 7 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | LANG.STRUCT.UCTCH  PARSE.MBDH | Masked by handler  Masked by default handler |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Uninitialized Memory** | STD-008-CPP | If the variable isn’t initialized before reading, it will create an indeterminate value. |

| **Noncompliant Code** |
| --- |
| Undefined behavior happens because “i” was not initialized first. There technically is nothing in “i”. |
| #include <iostream>    void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| Initialized “i” so that it has a value and memory |
| #include <iostream>    void f() {  int i = 0;  std::cout << i;  } |

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

| **Principles(s):** 3, 10 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 9.1p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local  Uninitialized variable |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **No Hard-Coding** | STD-009-CLG | We don’t want to hard-code sensitive information that might lead to injection attacks. |

| **Noncompliant Code** |
| --- |
| This hard-codes the const char\* and puts it into the function as a string, which could allow for many problems, especially when it allows them to authenticate anyway. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  if (!authenticate("correct code")) {  printf("Authentication error\n");  return -1;  }    printf("Authentication successful\n");  // ...Work with system...  return 0;  } |

| **Compliant Code** |
| --- |
| Asks for a security code; if it’s correct, allows them into the system; if it isn’t, destroys the data, and they have to try again. |
| int authenticate(const char\* code);    int main() {  #define CODE\_LEN 50  char code[CODE\_LEN];  printf("Please enter your authentication code:\n");  fgets(code, sizeof(code), stdin);  int flag = authenticate(code);  memset\_explicit(code, 0, sizeof(code));  if (!flag) {  printf("Access denied\n");  return -1;  }  printf("Access granted\n");  // ...Work with system...  return 0;  } |

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

| **Principles(s):** 1, 2, 3, 4, 5, 7, 8, 10 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2024.2 | CERT\_C-MSC41-a | Do not hard code string literals |
| Polyspace Bug Finder | R2025b | CERT C: Rule MSC41-C | Checks for hard coded sensitive data (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Logical Completeness | STD-010-CLG | This standard requires that all logic be considered when writing code. |

| **Noncompliant Code** |
| --- |
| Creates an if-else statement that does functions, but it does not account for anything but those two. This could cause an error if not only the system force quitting. |
| //Conditions being checked  if (a == b) {  /\* ... \*/  }  else if (a == c) {  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The code checks for logical completeness by adding an else clause that catches everything else that is not explicitly stated. This will also trigger an error that will catch everything else that we did not account for. |
| //All conditions are being checked because if the other two don’t work, then there is a catch error that handles everything else  if (a == b) {  /\* ... \*/  }  else if (a == c) {  /\* ... \*/  }  else {  /\* Handle error condition \*/  } |

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

| **Principles(s):** 2, 3, 10 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube C/C++ Plugin | 3.11 | ElseIfWithoutElse, SwitchWithoutDefault |  |
| Astrée | 24.04 | missing-else, switch-default | Partially 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.

Automation tools will help enforce standards. When we use automation tools, the goal is to test code to ensure it adheres to standards and catches errors that might otherwise go unnoticed. When using automation tools, we will use them in most of the pre-production phases. In the assess and plan stage of the DevOps process, we will start with what tools can be applied to what portion of the software. In the build phase, we will build out the automation, such as unit tests, so that they are implemented before the verify and testing phase. Lastly, the verification and testing phase will use those tools to ensure they follow our coding standards. Some tools we can use in pre-production include Cppcheck, unit testing, and Static Application Security Testing (SAST).

Now, in the production phase, we can implement monitoring and detection tools, such as Dynamic Application Security Testing (DAST), embedded in the application. We should also do some penetration testing to see how the code holds up to simulated attacks.

### 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-CLG | High | Unlikely | Medium | High | 2 |
| STD-002-CLG | Medium | Probable | Medium | Medium | 2 |
| STD-003-CPP | High | Probable | Medium | Medium | 2 |
| STD-004-JAV | High | Likely | High | High | 1 |
| STD-005-CPP | High | Likely | High | High | 1 |
| STD-006-CLG | Low | Unlikely | Low | Low | 3 |
| STD-007-CPP | Low | Probable | Low | Medium | 2 |
| STD-008-CPP | High | Probable | High | High | 1 |
| STD-009-CLG | High | Probable | High | High | 2 |
| STD-010-CLG | Medium | Probable | Medium | Low | 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** | **Encryption exists to secure data and prevent unauthorized access. This policy applies to any data used and aims to protect sensitive information by using locks to make it harder to access without the right permission.** |
| --- | --- |
| Encryption at rest | Encryption at rest is for data that is stored and not being used. When Encryption at rest is in place, use methods like XOR or AES, depending on the data's confidentiality. This protects the data from being accessed with the proper keys or permissions. |
| Encryption in flight | Encryption in flight is used for data sent to other network locations. This policy is important because it ensures that Green Pace is protected from interception attempts that could compromise its data. We do this by keeping the data encrypted and authenticating the endpoints that receive it. |
| Encryption in use | The encryption in use is for data currently being shared, and permission has been granted to read it. We do this by using authentication with a limited lifetime, so that when the time is up, the data is removed from the endpoint and the encryption key is destroyed, preventing continued access. |

| 1. **Triple-A Framework\*** | **Triple A Framework is for data permissions and information logging to ensure the software remains safe and to detect potential attacks. We do this by implementing authentication, authorization, and accounting.** |
| --- | --- |
| Authentication | Authentication allows us to verify that users and devices connected are trusted and authorized to interact with the software. We do this by password-protecting devices and information, using Multifactor Authentication (MFA) for user logins, and implementing timeouts. This is to protect the data and software from being compromised by an attacker. |
| Authorization | Authorization allows us to grant users permissions to access different types of information. User-level access is where we would implement the principle of least privilege, granting only what is strictly necessary. Each new user should be given the least amount of control possible, and there should be a default that denies access to sensitive information. This also involves changes being pushed to databases; they should be authorized and approved before the change takes place. |
| Accounting | Accounting logs all the users who touched data in the system. This is incredibly important when we are trying to defend against attacks or see the audit trail. This is one of the first warnings when attacked or data is being leaked unknowingly. Files accessed by users will be logged and through those logs we can find the users and find out what happened and to solve the issue. |

**\***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.] | 10/16/25 | Added fields | Hayley Blake |  |
| [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 |