**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 | Validating user input is to make sure that the user doesn’t input data that could be harmful or that could bypass security features, and this is done by restricting parameters. |
| 1. Heed Compiler Warnings | Heed Compiler Warnings has to do with allowing the compiler is using the highest warning level to make sure that you are able to find and fix all security issues. |
| 1. Architect and Design for Security Policies | Architect and Design for Security Policies is about planning the build out from the start and building the system into separate parts to make sure that users only have the amount of privileges that they should. |
| 1. Keep It Simple | Keeping it simple is as it sounds building the program as small and simple as possible and this is to make sure that there aren’t security flaws that cannot be seen because of how big and complex the system is. |
| 1. Default Deny | Default Deny is all about denying access to users who shouldn’t have access things that they shouldn't. |
| 1. Adhere to the Principle of Least Privilege | Adhere to the principle of least privilege is a lot like the last point which is to make sure that the user doesn’t have access to anything beyond what they need to use the service, and this is to make sure that little to no security issues can happen. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing Data is a lot like validating input data as you are making sure that data that could be harmful doesn’t make it to important data sets, so sanitizing that data and making sure that it doesn’t do any harm is important. |
| 1. Practice Defense in Depth | Defense in Depth is to make sure that you have several layers of defense as the chances of one layer failing is possible but if you have 6 or 7 layers that chance reduces a lot as a user needs to work around of lot of things just to gain access. |
| 1. Use Effective Quality Assurance Techniques | Effective QA is all about throwing as many testing strategies as possible such as penetration testing and all of this testing will make sure that if there is a flaw that you will find it. |
| 1. Adopt a Secure Coding Standard | Learn and use the Secure coding standard for your language and platform as every language and platforms will require different techniques. |

### 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-CCP] | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| Two different classes with the same name |
| class C{};  class C{}; |

| **Compliant Code** |
| --- |
| The two classes have two different names |
| class C{}:  class S{}; |

**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** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Type-compatibility definition-duplicate undefined-extern undefined-extern-pure-virtual external-file-spreading type-file-spreading | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60-a | - |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-DCL60-a | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| LDRA tool suite | 9.7.1 | 286 S, 287 S | Fully implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Does not read the uninitialized memory |

| **Noncompliant Code** |
| --- |
| Variable doesn’t have a value so there’s nothing for in the memory to read |
| Void f() {  Int I;  Std::cout<< I;  } |

| **Compliant Code** |
| --- |
| Variable now has a value and there for has something in the memory to be read |
| 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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Uninitialized-read | Partially checked |
| Helix QAC | 2021.2 | C++2726, C++2727, C++2728, C++2961, C++2962, C++2963, C++2966, C++2967, C++2968, C++2971, C++2972, C++2973, C++2976, C++2977, C++2978 | - |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Polyspace Bug Finder | R2021a | CERT C++: EXP53-CPP | Checks for: Non-initialized variable Non-initialized pointer Rule partially covered. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-nnn-CPP] | Don’t create a string with a null pointer |

| **Noncompliant Code** |
| --- |
| A string is created and when called the string returns a null and therefore leads to undefined behavior |
| Void f()  {  String tmp;  Std::cout << tmp;  } |

| **Compliant Code** |
| --- |
| String has a value and there returns something |
| Void f()  {  String c = “hello”;  Std::cout << c;  } |

**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** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert\_failure | - |
| Klocwork | 2021.1 | NPD.CHECK.CALL.MIGHT NPD.CHECK.CALL.MUST NPD.CHECK.MIGHT NPD.CHECK.MUST NPD.CONST.CALL NPD.CONST.DEREF NPD.FUNC.CALL.MIGHT NPD.FUNC.CALL.MUST NPD.FUNC.MIGHT NPD.FUNC.MUST NPD.GEN.CALL.MIGHT NPD.GEN.CALL.MUST NPD.GEN.MIGHT NPD.GEN.MUST RNPD.CALL RNPD.DEREF | - |
| Helix QAC | 2021.2 | C++ 4770, C++ 4771, C++ 4772, C++ 4773, C++ 4774 | - |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| User input isn’t checked so they are able to input certain things and inject code |
| SQL = getRequestString("username");  SQLU = getRequestString("userpassword");  sql = “SELECT \* FROM Users WHERE Name = " + SQL + " AND Pass = " + SQLU + ” |

| **Compliant Code** |
| --- |
| User input is check and therefore isn’t alllowed to inject their own code |
| PreparedStatement pStmt = PreparedStatement();  std::cin >> username;  std::cin >> userpassword;  sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;”, username, userpassword}; |

**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** |
| --- | --- | --- | --- | --- |
| High | probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | SQL\_PREPARED\_STATEMENT\_GENERATED\_FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| The Checker FrameWork | 2.1.3 | Tainting Checker | Trust and security errors |
| Fortify | 1.0 | HTTP\_Response\_Splitting SQL\_Injection\_Persistence SQL\_Injection | Implemented |
| Parasoft Jtest | 2021.1 | CERT.IDS00.TDSQL | Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Doesn’t access freed memory |

| **Noncompliant Code** |
| --- |
| After the memory has been deallocated it becomes a write-after-free and therefore can be written to if the correct permissions are held. |
| Struct S  {  void f();  }  void h()  noexcept(false) {  S \* s = new S; |

| **Compliant Code** |
| --- |
| The memory is not deallocated until need |
| Struct S  {  void f();  }  void h()  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):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Parasoft insure++ | - | - | Runtime detection |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| The assert isn’t tested |
| 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** |
| --- |
| The constant expression is tested |
| struct timer {  unsigned **char** MODE;  unsigned **int** DATA;  unsigned **int** COUNT;  };  (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int))) |

**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 | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | - |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemeted |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Terminating the program without warning |

| **Noncompliant Code** |
| --- |
| A glitch may happen where terminate() may be called and end the program |
| void term\_func()  noexcept(false);  void f() {  term\_func();  }  **int** main() {  if (0 != std::**atexit**(f)) {  }  } |

| **Compliant Code** |
| --- |
| If f is called this time then the error is caught and doesn’t break the program |
| void term\_func()  noexcept(false);  void f() {  try { term\_func();  }  catch (...)  {  }  **int** main() {  if (0 != std::**atexit**(f)) {  } |

**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 | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1Po | BADFUNC.ABORT BADFUNC.EXIT | USe of abort Use of exit |
| Klocwork | 2021.1 | MISRA.CATCH.ALL CERT.ERR.ABRUPT\_TERM | - |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |
| Polyspace Bug Finder | R2021a | CERT C++: ERR50-CPP | Checks for implicit call to terminate() function (rule partially covered |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| OOP | [STD-008-CPP] | OOP can cause issues with security if things happen out of order |

| **Noncompliant Code** |
| --- |
| The order is wrong so value aren’t specified and therefore will cause issues |
| Class C {  Int a;  Int b;  Public: C(int d) : b(d), b(a + 1) {}  }; |

| **Compliant Code** |
| --- |
| The order has been fixed so values are specified and will work correctly |
| Class C {  Int a;  Int b;  Public C (int d) : a(d), b(a + 1) {}  }; |

**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** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Initializer-list-order | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP53 | - |
| LDRA | 9.7.1 | 206 S | Fully implemented |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-OOP53-a | List members in an initialization list in the order in which they are declared |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | [STD-009-CPP] | Accesses an object outside of its lifetime |

| **Noncompliant Code** |
| --- |
| A pointer is called before the pointers lifetime and therefore results in an undefined behavior |
| struct S {  void mem\_fn();  };  void f() {  S \*s;  s->mem\_fn();  } |

| **Compliant Code** |
| --- |
| The pointer is now called after the lifetime |
| struct S {  void mem\_fn();  };  void f() {  S \*s = new S;  s->mem\_fn();  delete s;  } |

**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** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Return-reference-local dangling\_pointer\_use | Partially checked |
| Clang | 3.9 | -Wdangling\_initializer-list | Catches sine lifetime issues related to incorrect use of std::initializer\_list<> |
| CodeSonar | 6.1p0 | IO.UAC ALLOC.UAF | Use after close Use after free |
| Parasoft C/C++ | 2021.1 | CERT\_CPP-EXP54-a  CERT\_CPP-EXP54-b  CERT\_CPP-EXP54-c | Do not use resources that have bneen freed the address of an object with automatic storage shall not be returned from a function The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has creased to exist |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Contatiners | [STD-010-CPP] | Use valid iterator ranges |

| **Noncompliant Code** |
| --- |
| On each iteration of its internal loop, std::for\_each() compares the first iterator (after incrementing it) with the second for equality; as long as they are not equal, it will continue to increment the first iterator. Incrementing the iterator representing the past-the-end element of the range results in undefined behavior. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.end(), c.begin(), [](int i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| The iterator values passed to std::for\_each() are passed in the proper order. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.begin(), c.end(), [](int i) { std::cout << i; });  } |

**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** |
| --- | --- | --- | --- | --- |
| High | Probable | High | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Overflow\_upon\_dereference | - |
| Helix QAC | 2021.2 | C++ 3802 | - |
| Parasoft C/C++ | 2021.1 | CERT\_CPP-CTR53-a CERT\_CPP-CTR53-b | Do not use an iterator range that isn’t really a range Do not compare iterators from different containers |
| PVS-Studio | 7.14 | V539, V662, V789 | - |

### 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.

Everything starts out in Pre-production, everyone gets together and starts planning the landscape of the threats that are currently happening and have happened in the past, while creating a backlog for future threats that may happen in the future, after the planning phase starts the designing phase which is where the developers and security teams start implementing the backend security, after the designing phase has ended and everything has been developed up the standards it is time for the team to start building and running the system and making sure everything works correctly and if everything works correctly then they start verifying and testing both the software and the security of that software, after the software has entered production what happens for the security team now is that they need to maintain and stabilize any security flaws within the software and respond to any attacks or threats that do happen to bypass the security or has a chance to, but most of their job from there on out is to monitor and detect any new threats to the system.

### 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 | Probable | Medium | High | 1 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Probable | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | low | Probable | Medium | low | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | Medium | 3 |
| STD-009-CPP | High | Probable | High | High | 2 |
| STD-010-CPP | High | Probable | High | High | 2 |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption in rest protects stored data. This may include hard drives, phones, computers, and cloud assets, among others. Protection of this data can be done through encryption tools, disk encryption and security for mobile devices and computers. |
| Encryption at flight | Encryption at flight is about protecting data that is moving. This can be between two devices within a network, or moving outside of a network. This can be protected through examples such as email encryption, DLP solutions, and solid network security features, such as firewalls and authentication. It is also important to consider the path data may be taking, and the security of this path. |
| Encryption in use | Encryption in use protects data that is created, edited, or otherwise defined as in-use. Protection of this data can be done by ensuring data control and protection exists prior to use, and in place in the first place. Managing access rights and identity will also minimize risk to this data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the act of confirming one’s identity. This can cover a range of types, but often are examples such as static passwords, one-time passwords, certifications, and biometric credentials. These forms of identification work to ensure a person is who they claim to be. |
| Authorization | Authorization specifies the access rights and privileges of a user, and are an important part of information and computer security. Where authentication confirms an identity, authorization determines what a user can and cannot access in the first place, limiting possible vulnerabilities when someone may interact with sensitive data they may not need to access, or the permissions one may have during access. |
| Accounting | Accounting is the process of keeping track of activity while interacting with a system, showing timestamps, accessed resources, and data transfer information. This is valuable in both creating a “bread crumb trail” in user activity, and also for the purposes of forensic analysis and investigation, should it be required. |

**\***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 |  |
| 1.5 | 05/19/2023 | First Project | Dusten Schacht | Olga Mill |
| 2.0 | 06/10/2023 | Finished Project | Dusten Schacht | Olga Mill |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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