**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 | By validating input data, we can ensure that the data being passed through the system is valid input and we can ensure that the data being passed is what we are looking for. This validation can also ensure that the input isn’t a malicious string that could cause issues in the data systems that the code is connected to. |
| 1. Heed Compiler Warnings | While compiling the warnings can show errors or small issues in the code that could lead to larger issues as the code gets more complex or runs longer. This could be something as simple as a minor memory leak that could cause the program to appear to be running ok, but after a extend run time can fail and crash. By heeding compiler warnings, we can review the code and find/fix the issues or design a better method for performing the functions so that the code does not have these issues. |
| 1. Architect and Design for Security Policies | By ensuring that the design and architecture of the programs meet security policies, it will ensure that the code being developed will not violate these policies. Having these policies and maintaining these will ensure that the software and data collected, maintained, and developed by the company will be secure and safe. |
| 1. Keep It Simple | Keeping the code simple can allow for it to be easier to review. This will also allow for some errors to be easier to find. By keeping the overall code for the program simple there is less chances for there to be unnecessary redundant code that are running at the same time that could cause the program to not function properly or could slow down the program. |
| 1. Default Deny | If the input or connection does not meet appropriate requirements having a default denial, will allow for the system to be maintained secure. If no data is checked for its validity or that it is correct, it could cause issues with the data structure that it is maintained in. This could also allow for unintended access to systems. |
| 1. Adhere to the Principle of Least Privilege | By ensuring that users have appropriate level of access ensure that the data is maintained as secure as possible. Maintaining access to specific data only to those who are working with or on this data will ensure that the data is not affected by others, and it will also prevent the data from being access by individuals who could use that data inappropriately. |
| 1. Sanitize Data Sent to Other Systems | By ensuring that the data being sent to other systems, is only the data that those systems will uses will ensure that the size of data transfer will be smaller. It will allow for those systems to read and use that data quicker since it will not need to pull only the information that it needs. This will also ensure that malicious data isn’t sent to these systems that could cause issues or errors to occur in these systems |
| 1. Practice Defense in Depth | Having multiple levels of security defenses allows for the systems and data being stored in these systems to be maintained safe. This will help to ensure data protection and minimize the chances of data corruption. By having multiple layers of defense and multiple avenues to ensure defense, will allow for the systems to not rely only a single point that could fail. If a system or program only relies on a single defense point it would mean only that point needs to fail and the whole system and all its data will be left unsecure. |
| 1. Use Effective Quality Assurance Techniques | By performing quality assurance techniques, we can ensure that the code being developed and released is clean and secure. This means that the code and programs being released will perform appropriately and will have appropriate checks implemented in it to ensure that the data being passed through the program should have no issues. This means that all data being received from users is validated and that the information being sent to other systems is meant for those systems and does not have any unnecessary information. This will ensure that the code will interact smoothly with all connected systems and programs. |
| 1. Adopt a Secure Coding Standard | Practicing secure coding will ensure that the code being used will not open the system to unauthorized access. This will ensure that the data being used by the programs developed and the systems they are accessing is not released to unintended parties. By adopting a secure coding standard it will ensure that all developers are meeting these requirements and will allow for the developers to have the proper basic requirements necessary to perform the tasks they are asked to complete. |

### 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] | Correct data type for intended use and has the appropriate space required. |

| **Noncompliant Code** |
| --- |
| Though a string can be input into the char data type if data type has an incorrect length or data input exceeds defined limit could lead to overflow |
| Void input (){  char input[20];  std::cin >> input;  } |

| **Compliant Code** |
| --- |
| If data type is chosen that will allow for all expected input data is used overflow will be prevented. |
| Void input(){  Std:string input;  Std::cin >> input;  } |

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

| **Principles(s):** This falls under the Valid Data input. If the structure being used to retain the data input by the user then data that is entered could be lost or could cause the program to fail to run or crash. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | LANG.MEM.BO  LANG.MEM.TO | Buffer overrun  Type Overrun |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Ensure that the values being put into data do not exceed data type limit |

| **Noncompliant Code** |
| --- |
| If a limit is set for values and there is nothing limiting user input a overflow could occur. |
| Void m (){  char input[20];  std::cin << input;  } |

| **Compliant Code** |
| --- |
| By setting a max value and getting user input will referencing the max value it will help prevent a buffer overflow |
| Void m(){  Const size\_t max = 21;  Char input[max +1];  Std::cin.getline(input,max,’\n’);  Cout << output;  } |

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

| **Principles(s):** Valid data input. With the since of the variable being set it ensure that the program will run, and that the data entered is retained. That being said there should be proper warnings on how much data can be input into the program so nothing is lost. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | LANG.MEM.BO  LANG.MEM.TO | Buffer overrun  Type Overrun |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-03-CPP] | Do not Attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| By modifying a string literal it is performing an undefined action |
| Char \*str = “string literal”;  Str[0] = ‘S’ |

| **Compliant Code** |
| --- |
| If char is initialized as an array. The string can be safely modified based on array location |
| Char str[] = “string literal”;  Str[0] = ‘S’; |

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

| **Principles(s):** Use Effective Quality Assurance Techniques. By not ensuring that the modifications are done properly could causes issues with the programs running and could cause unexpected crashing of the program. By ensuring that the appropriate quality assurance techniques would ensure that these problems do not exist. This issue could also lead to security issues with the program as well. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2021.2 | CERT\_C-STR30-a  CERT\_C-STR30-b | String literal shall not be modified  Do not modify string literals |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Check for or statements in or query to prevent injections |

| **Noncompliant Code** |
| --- |
| Not performing a SQL string check for added statements such as “or” statements could lead to SQL injection |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  records.clear();  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  }  void run\_queries(sqlite3\* db)  {  char\* error\_message = NULL;  std::vector< user\_record > records;  // query all  std::string sql = "SELECT \* from USERS";  if (!run\_query(db, sql, records)) return;  dump\_results(sql, records);  // query 1  sql = "SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred'";  if (!run\_query(db, sql, records)) return;  dump\_results(sql, records);  // run query 1 with injection 5 times  for (auto i = 0; i < 5; ++i)  {  if (!run\_query\_injection(db, sql, records)) continue;  dump\_results(sql, records);  }  } |

| **Compliant Code** |
| --- |
| By adding a check for for a string statement of either “ or “ or “ OR “, will allow for the program to see if there is a break sql search command that could be trying to obtain other data. |
| bool run\_query(sqlite3\* db, const std::string& sql, std::vector< user\_record >& records)  {  // TODO: Fix this method to fail and display an error if there is a suspected SQL Injection  // NOTE: You cannot just flag 1=1 as an error, since 2=2 will work just as well. You need  // something more generic  std::string found = " or "; //lower case search string  std::string found2 = " OR "; //upper case search string  // Checking sql statement for lower case or statement to return false  if (sql.find(found) != std::string::npos) {  return false;  }  // Checkign sql statement for upper case or statement to return false  if (sql.find(found2) != std::string::npos) {  return false;  }  records.clear();  char\* error\_message;  if(sqlite3\_exec(db, sql.c\_str(), callback, &records, &error\_message) != SQLITE\_OK)  {  std::cout << "Data failed to be queried from USERS table. ERROR = " << error\_message << std::endl;  sqlite3\_free(error\_message);  return false;  }  return true;  }  void run\_queries(sqlite3\* db)  {  char\* error\_message = NULL;  std::vector< user\_record > records;  // query all  std::string sql = "SELECT \* from USERS";  if (!run\_query(db, sql, records)) return;  dump\_results(sql, records);  // query 1  sql = "SELECT ID, NAME, PASSWORD FROM USERS WHERE NAME='Fred'";  if (!run\_query(db, sql, records)) return;  dump\_results(sql, records);  // run query 1 with injection 5 times  for (auto i = 0; i < 5; ++i)  {  if (!run\_query\_injection(db, sql, records)) continue;  dump\_results(sql, records);  }  } |

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

| **Principles(s):** Adopt a secure coding standard. By ensuring that the code being entered is appropriate for the software ensure that the program will not perform any actions that the user is not supposed to perform. In this case where a sql injection is prevented, it ensures that a user is unable to perform actions in the database that are not desired by the designers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft SOAtest | 2021.2 | Penetration Test | Penetration Test used to check for SQL Injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly use sharded Pointers to ensure |

| **Noncompliant Code** |
| --- |
| Storing the same pointer into two different pointers |
| Void f(){  Int \*I = new int;  Std::shared\_ptr<int> p1(i);  Std::shared\_ptr<int> p2(i)  } |

| **Compliant Code** |
| --- |
| Use the make\_shared call to correctly share pointer data. This will allow for the 2 pointers to be related and no errors will occur |
| Void f(){  Std::shared\_ptr<int> p1 = std::make\_shared<int>();  Std::shared\_ptr<int>p2(p1);  } |

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

| **Principles(s):** Secure coding. By not properly setting pointers it will create vulnerabilities in the code that can be exploited. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2021.2 | **CERT\_CPP-MEM56-a** | Do not store an already-owned pointer value in an unrelated smart pointer |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use assert appropriately |

| **Noncompliant Code** |
| --- |
| If num1 does not equal to 1 the follow on code will not execute and will terminate |
| Assert(num1 == 1) |

| **Compliant Code** |
| --- |
| As long as num1 has a value other than NULL code will proceed. |
| Assert(num1 != NULL) |

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

| **Principles(s):** Quality Assurance Techniques. By ensuring that assertions are not used in a way that is inappropriate it will ensure that the program runs and runs appropriately. With how the code is shown above in the un compliant code. The program will terminate when ever num1 doesn’t equal 1 |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exceptions must be handled |

| **Noncompliant Code** |
| --- |
| Std::terminate will be called if no function is handling the throwing\_func |
| Void throwing\_func() noecept(false:;  Void f () {  Throw\_func();  }  Int main() {  F();  } |

| **Compliant Code** |
| --- |
| The main function will handle the throw\_func and handle the error |
| Void throwing\_func() noecept(false:;  Void f () {  Throw\_func();  }  Int main() {  Try{  f();  }catch(…){  error  }  } |

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

| **Principles(s):** Valid Quality assurance techniques. All errors need to be handle in order to ensure that program does not terminate unexpectedly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2021.2 | Cert\_CPP-ERR51-a  CERT\_CPP-ERR51-b | always catch exceptions |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | [STD-008-CPP] | Handling exceptions prior to main |

| **Noncompliant Code** |
| --- |
| Exception thrown but not caught when globalS is constructed |
| Struct S {  S() noexcept(false);  };  Static S globalS; |

| **Compliant Code** |
| --- |
| Exceptions will be caught during the object construction since the constructor will be called during globalS |
| Struct S{  S() noexcept();  };  S &globalS(){  Try {  Static S s;  Return s;  } catch(…){  //error will be handled  }  } |

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

| **Principles(s):** Valid Quality Assurance techniques and Secure coding. By Ensuring that errors are handled prior to main and handled properly will ensure that program does not terminate unexpectedly and does not create vulnerabilities in the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ | 2021.2 | CERT\_CPP-ERR58-a | Exception shall be raised only after start-up and before termination of the program. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Memory | [STD-009-CPP] | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| S is dereferenced after it was freed. |
| Struct S{  Void f();  };  Void g() noexcept(false){  S \*s = new S;  Delete s;  s-> f();  } |

| **Compliant Code** |
| --- |
| S not deallocated until it is dereferenced to f() |
| 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):** Secure coding. By accessing freed memory it could allow for a vulnerability to be created allowing unwanted code to be ran in that instance. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | ALLOC.UAF | Use after Free |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| exception | [STD-010-CPP] | Detect errors when converting |

| **Noncompliant Code** |
| --- |
| No check is performed to see if input are numeric values |
| Void f(){  Int I, j;  Std::cin >> I >> j;  } |

| **Compliant Code** |
| --- |
| Input checked for correct data type and throws exceptions |
| Void f(){  Int I, j;  Std:: cin.exceptions(std::istream::failbit | std::istream::badbit);  Try {  Std:: cin>> I >> j;  } catch (std::istream::failure &failure &E){    }  } |

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

| **Principles(s):** Secure coding. If used in a program that requires high security requirements this could lead to a vulnerability. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | BADFUNC.ATOF | Use atof  Use atoll  Use atoll |

### 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.

Some standards can be using during the design process to ensure that assertions are performed in the appropriate locations. During the verify and test process Error handling can be verified and ensure that the program is running properly, and all automate checks can be verified that the program meets the correct requirements. During the Transition and health check during the production phase penetration testing can be performed to ensure that the code that requires access to the SQL database is not allowing inappropriate queries are being ran.

### 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 | Likely | Medium | High | 2 |
| STD-002-CPP | High | Likely | Medium | High | 2 |
| STD-003-CPP | Low | Likely | Low | Medium | 2 |
| STD-004-CPP | High | Likely | High | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Low | Low | Low | 3 |
| STD-007-CPP | Low | Likely | Medium | Low | 3 |
| STD-008-CPP | Low | Likely | Low | Medium | 2 |
| STD-009-CPP | High | Likely | Medium | High | 1 |
| STD-010-CPP | Medium | Unlikely | Medium | Medium | 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 is the encryption of the data that is stored on the hard drives and not in use. By ensuring that the data the is currently not being used by the program is encrypted, ensures that if the data is access outside of the program, it is still encrypted, and it will be harder to use/read with out the appropriate keys. |
| Encryption at flight | Encryption at flight, is the encryption of the data that is being transmitted across the network. By ensuring that all data being transmitted across the network from a program it will ensure that unauthorized access to this data is minimized. Though the data could possibly be obtained it would be harder to use/read. |
| Encryption in use | Encryption in use ensures that the data always remains encrypted. Since we would already have the data encrypted when at rest or in flight. By also encrypting the data while in use it allows for the data to be maintained as secure as possible. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication ensures that the user/programs being ran on a system are recognized by the system and are allowed to have access to the system. |
| Authorization | Authorization ensures that the user/program has the appropriate privilege to access specific areas/data of the system. This will ensure that the system is secure, and the principle of least privilege is upheld |
| Accounting | Accounting is used to monitor and track the data that is accessed and how it is used on the network/system. This is necessary when programs are being ran that access secure data. This will ensure data security and help track any unauthorized access to this information. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 04/10/2022 | Update | Jacob Weidner |  |
| [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 |