**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 | Data validation is confirming that the input given to the program meets a certain set of requirements. These requirements are defined by the developer when writing the code. |
| 1. Heed Compiler Warnings | During the development of any piece of code the compiler will generate warnings for the developer. These warnings often tell the developer of possible shortcomings in their code that could lead to errors. |
| 1. Architect and Design for Security Policies | This principle is the act of integrating security policies in to the architecture and design phase of a program. Threat modeling is a great way of understanding the weaknesses of a system and implementing a solution in the design phase. |
| 1. Keep It Simple | The more complex a program the more vulnerabilities it could have, and quite possibly the more points to defend. The simpler the system is the easier it is to understand, which makes it that much easier to defend. |
| 1. Default Deny | Default deny is the act of prohibiting all access or permissions unless explicitly stated. What this does is limit possible avenues for the attacker to exploit. |
| 1. Adhere to the Principle of Least Privilege | This principal is very similar to default deny. Least privilege is only granting access to things necessary to complete a given task. Limiting access like this reduces the chance of misuse of the system. |
| 1. Sanitize Data Sent to Other Systems | Data sanitization is the act of cleaning up the given data by stripping any unnecessary aspects. Along with discarding extra data, sanitization is also the act validating input data and that it conforms to a specific format. |
| 1. Practice Defense in Depth | Defense in depth is the implementation of a series or layers of security measures. The more road blocks that impede an attacker the harder it will be for them to gain access. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance doesn’t mean make sure the program works as intended before it is delivered to the customer. Ensuring quality assurance also means confirming that the software is secure and free from defects and any fixable vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Adopting a coding standard ensures that all code written is consistent among all the development teams. |

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Do not modify the standard namespaces |

[DCL58-CPP. Do not modify the standard namespaces - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/DCL58-CPP.+Do+not+modify+the+standard+namespaces)

| **Noncompliant Code** |
| --- |
| Example of declaration of a new variable in the standard namespace, which would follow with an undefined behavior. |
| namespace std {  float y;  } |

| **Compliant Code** |
| --- |
| Compliant code declares a variable into a non-standard namespace, and thus avoiding problems with identifiers. |
| namespace xyz {  float y;  } |

| **Principles(s):** Use Effective Quality Assurance Techniques: Ensuring that the code does not result in an undefined behavior the principle of effective quality assurance is being upheld. If the code works as intended and is free of any bugs that cause something unplanned a level of quality is being held. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.STRUCT.DECL.SNM | Static Code Analysis Tool |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-DLC58A | IDE Embedded Analysis Tool |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Do not depend on the order of evaluation for side effects. |

[EXP50-CPP. Do not depend on the order of evaluation for side effects - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP50-CPP.+Do+not+depend+on+the+order+of+evaluation+for+side+effects)

| **Noncompliant Code** |
| --- |
| This noncompliant code is using an overloaded operator which operates in the same manner as a function call. The operands aren’t being evaluated in the typical left to right manner, but are actually un-sequenced. |
| int main(){  std::cout << x-- << ‘ ‘ << x << std::endl;  } |

| **Compliant Code** |
| --- |
| The way that this code is laid out there is a clear sequence in how the program is supposed to be executed. |
| int main(){  std::cout << x-- << std::endl;  std::cout << x << std::endl;  } |

| **Principles(s):** Architect and Design for Security Policies: Following this Coding standard ensures that the program is free of the undefined behavior that comes by not following this standard. Undefined behavior is unknown and therefor not secure which could cause a potential vulnerability. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.STRUCT.SE.DEC  LANG.STRUCT.SE.INC | Static Code Analysis Tool |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-EXP50-a  CERT\_CPP-EXP50-b  CERT\_CPP-EXP50-c  CERT\_CPP-EXP50-d  CERT\_CPP-EXP50-e  CERT\_CPP-EXP50-f | IDE Embedded Analysis Tool |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CLG | Guarantee that storage for strings has sufficient space for character data and the null terminator. |

[STR31-C. Guarantee that storage for strings has sufficient space for character data and the null terminator - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/STR31-C.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator)

| **Noncompliant Code** |
| --- |
| This non compliant code takes input and stores it in to a character array with a size of 10. If the input is more than 10 characters this would result in a buffer overflow. |
| void func(){  char input[10];  std::cin >> input;  } |

| **Compliant Code** |
| --- |
| The code below prevents the buffer over flow by changing the character array to a standard string. |
| void func(){  std::string input;  std::cin >> input;  } |

| **Principles(s):** Validate Input Data, and Architect and design for security policies: Following this rule mitigates the chance of buffer overflow from happening which could result in a security breach. Stopping the overflow allows for the program to validate the data that was input. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.MEM.BO  LANG.MEM.TO  MISC.MEM.NTERM  BADFUNC.BO.\* | Static Code Analysis Tool |
| Parasoft C/C++test | 2023.1 | CERT\_C-STR31-a  CERT\_C-STR31-b  CERT\_C-STR31-c  CERT\_C-STR31-d  CERT\_C-STR31-e | IDE Embedded Analysis Tool |

#### Coding Standard 4

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

[IDS00-J. Prevent SQL injection - SEI CERT Oracle Coding Standard for Java - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection)

| **Noncompliant Code** |
| --- |
| This noncompliant code fails to validate or sanitize the input data. |
| Scanner scanner = new Scanner(System.in);  userID = scanner.nextLine();  password = scanner.nextLine();  Sql = “SELECT \* FROM Users WHERE Name = “ + userID + “ AND Pass = “ + password; |

| **Compliant Code** |
| --- |
| This block of code utilizes the PreparedStatment class that enforces type checking. This ensures that the passed parameters are of the correct type. |
| Scanner scanner = new Scanner(System.in);  userID = scanner.nextLine();  password = scanner.nextLine();  SQL\_statment = “SELECT \* FROM Users WHERE Name = ? AND Password = ?”  PreparedStatement x = connection.prepareStatment(SQL\_statment);  X.setstring(1, userID);  x.setstring(2, password);  x.executeQuery(); |

| **Principles(s):** Validate Input Data: This standard upholds the principal by enforcing the developer to ensure the input data is validated and sanitized. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | JAVA.IO.INJ.SQL | Static Code Analysis Tool |
| Parasoft Jtest | 2023.1 | CERT.IDS00.TDSQL | IDE Embedded Analysis Tool |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Do not store an already-owned pointer value in an unrelated smart pointer. |

[MEM56-CPP. Do not store an already-owned pointer value in an unrelated smart pointer - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM56-CPP.+Do+not+store+an+already-owned+pointer+value+in+an+unrelated+smart+pointer)

| **Noncompliant Code** |
| --- |
| This noncompliant code assigns two pointers to a local variable. This would cause a problem if one of the two smart pointers were deleted, because it would also delete the local variable that the other smart pointer is also assigned. |
| String \*x = new string;  Std::shared\_ptr<string> y(x);  Std::shared\_ptr<string> z(x); |

| **Compliant Code** |
| --- |
| The two shared pointers are made by copying each other and thus related. |
| Std::shared\_ptr<string> y = std::make\_shared<string>();  Std::shared\_ptr<string> z(y); |

| **Principles(s):** Architect and Design for Security Policies: This Standard upholds this principal by preventing something that results in an undefined behavior that could possibly lead to exploited vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MEM56-a | IDE Embedded Analysis Tool |

#### Coding Standard 6

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

[MSC11-C. Incorporate diagnostic tests using assertions - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/MSC11-C.+Incorporate+diagnostic+tests+using+assertions#:~:text=MSC11-C.%20Incorporate%20diagnostic%20tests%20using%20assertions%201%20Noncompliant,software%20defects%20that%20may%20result%20in%20vulnerabilities.%20)

| **Noncompliant Code** |
| --- |
| This code block fails to use any diagnostic testing using assertions. |
| int divide(int a, int b) {  return a / b;  }  int main() {  divide(10, 2) == 5);  divide(10, 0)); // Triggers a failure  return 0;  } |

| **Compliant Code** |
| --- |
| This example uses the assert macro to check if b is not equal to zero before performing division. |
| int divide(int a, int b) {  assert(b != 0 && "Division by zero");  return a / b;  }  int main() {  // Test divide function  assert(divide(10, 2) == 5);  assert(divide(10, 0)); // Triggers an assertion failure  return 0;  } |

| **Principles(s):** Use Efffective Quality Assurance Techniques: One way to ensure quality is by performing testing. This standard shows the use of assertion testing. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.FUNCS.ASSERTS | Static Code Analysis Tool |
| Parasoft C/C++test | 2023.1 | CERT\_C-MSC11-a | IDE Embedded Analysis Tool |

Coding Standard 7

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

[ERR51-CPP. Handle all exceptions - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions)

| **Noncompliant Code** |
| --- |
| This code block doesn’t have a handler to address an exception being thrown. |
| int divide(int a, int b) {  if (b == 0) {  throw runtime\_error("Division by zero");  }  return a / b;  }  int main() {  cout << divide(10, 0) << endl;  return 0;  } |

| **Compliant Code** |
| --- |
| This code block implements a try / catch block in the main function. This statement handles the error thrown by the divide function when the variable b is equal to zero. |
| int divide(int a, int b) {  if (b == 0) {  throw runtime\_error("Division by zero");  }  return a / b;  }  int main() {  try{  cout << divide(10, 0) << endl;  } catch (const exception& x) {  cerr << “Exception : “ << x.what() << endl;  }  return 0;  } |

| **Principles(s):** Architect and Design for Security Policies: This standard ensures that all exceptions are handled so the program doesn’t terminate randomly. Stopping this from happening increases security by mitigating denial of service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.STRUCT.UCTCH | Static Code Analysis Tool |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR51-a  CERT\_CPP\_ERR51-b | IDE Embedded Analysis Tool |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Miscellaneous** | STD-008-CPP | Value returning functions must return a value from all exit paths |

[MSC52-CPP. Value-returning functions must return a value from all exit paths - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MSC52-CPP.+Value-returning+functions+must+return+a+value+from+all+exit+paths)

| **Noncompliant Code** |
| --- |
| This code block is missing a way for the function to return a value in the event the if statement is false. |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| This code block has a return avenue for the if statement as well as the default return. Which means the function will always return a int value. |
| Int absolute\_value (int a, int b) {  If (a < b) {  return a;  }  return b;  } |

| **Principles(s):** Architect and Design for Security Policies: This Standard upholds this principal by preventing something that results in an undefined behavior that could possibly lead to exploited data vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | LANG.STRUCT.MRS | Static Code Analysis Tool |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-MSC52-a | IDE Embedded Analysis Tool |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Expressions** | STD-009-CPP | Do not access an object outside of its lifetime. |

[EXP54-CPP. Do not access an object outside of its lifetime - SEI CERT C++ Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP54-CPP.+Do+not+access+an+object+outside+of+its+lifetime)

| **Noncompliant Code** |
| --- |
| The code below shows a new pointer being made and then deleted. Then the same pointer is used to call a member function after the storage for the pointer had been deallocated. |
| struct Example {  void returnTen();  };  Void function() {  Example \*a = new Example;  delete a;  a -> returnTen();  } |

| **Compliant Code** |
| --- |
| The following code is compliant because the only calls made to the pointer object are during its lifetime while there is storage allocated for the object. |
| struct Example {  void returnTen();  };  Void function() {  Example \*a = new Example;  a -> returnTen();  delete a;  } |

| **Principles(s):** Architect and Design for Security Policies: This Standard upholds this principal ensuring an attacker doesn’t have the chance to randomly run any of the code. Thus increasing Security of the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | IO.UAC  ALLOC.UAF | Static Code Analysis Tool |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-EXP54-a  CERT\_CPP-EXP54-b  CERT\_CPP-EXP54-c | IDE Embedded Analysis Tool |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Integers** | STD-010-CLG | Ensure that operations on signed integers do not result in overflow. |

[INT32-C. Ensure that operations on signed integers do not result in overflow - SEI CERT C Coding Standard - Confluence (cmu.edu)](https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow)

| **Noncompliant Code** |
| --- |
| This non compliant code uses the + operator, and could result in an overflow. |
| int add(signed int x, signed int y){  signed int result = x + y;  } |

| **Compliant Code** |
| --- |
| The compliant code takes the argument values and runs them through a check using the || and && operators. The if else statement that was implemented catches the overflow and handles the error that follows. |
| int add(signed int x, signed int y){  signed int result;  if (((y > 0) && (x < (INT\_MIN – y)) || ((y < 0) && (x > (INT\_MAX – y)))) {  //throws error  }  else {  result = x + y;  }  Return result;  } |

| **Principles(s):** Architect and Design for Security Policies: This Standard upholds this principal by preventing buffer overflows. Buffer Overflows can allow an attacker the chance to randomly run pieces of the code. Thus increasing Security of the program. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 8.0p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  MIS.MEM.SIZE.ADDOFLOW  MISC.MEM.SIZE.BAD  MISC.MEM.SIZE.MULOFLOW  MISC.MEM.SIZE.SUBUFLOW | Static Code Analysis Tool |
| Parasoft C/C++test | 2023.1 | CERT\_C-INT32-a  CERT\_C-INT32-b  CERT\_C-INT32-c | IDE Embedded Analysis Tool |

### 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.

I feel that the optimal point to integrate automation would be on the pre-production side. Specifically in the Build section as the automation that I have suggested work by alerting the developer of the issue during and after a build.

### 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 | Medium | 2 |
| STD-002-CPP | Medium | Probable | Medium | Medium | 2 |
| STD-003-CLG | High | Likely | Medium | High | 1 |
| STD-004-JAV | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CLG | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | Medium | Probable | Medium | Medium | 2 |
| STD-009-CPP | High | Probable | High | High | 1 |
| STD-010-CLG | High | Likely | High | High | 1 |

### 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 protection of data while being stored on a variety of devices as well as cloud storage locations. The encryption happens as the data is being written to the end device with an encryption algorithm. One of the best algorithms to use is the AES-256 algorithm. This algorithm changes the data based on the value of the algorithm. The only way to read the data is to have the key that allows for easy translation back in to its original format. This Policy applies as soon as data is being written to the drive or being accessed. |
| Encryption at flight | This type of encryption is being done while the data is in transit between two points. Since the data is encrypted when it is sent the only way to read it would be to have the correct key to translate the data accurately. |
| Encryption in use | Encryption in use is when data is being encrypted as it is being used by an application. This type of encryption helps mitigate unauthorized access while the data is being used. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is verifying the identity of the user trying to access the system, and is most commonly done with a username and password. This policy applies since it prohibits access to the system to anyone that does not have the proper credentials to login. |
| Authorization | Authorization is the act of limiting access to certain elements of a system unless authorized. This could be in the form of limiting the data the user can see, or actions they are allowed to perform. Regular users might be able to add or make changes to the database, but an admin has the ability to create new users and create access levels. Using authorization like this allows the owner to structure user roles and access so they uphold the principal of least privilege. |
| Accounting | Once a user logs in to a system the act of creating logs and transactions for everything the user does is called accounting. Accounting allows the ability for user activity to be watched as well as potentially stop misuse / loss of resources. |

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 | 01/28/2024 | Principals and Coding Standards updated | Robert Golden |  |
| 1.2 | 2/16/2024 | Update to entire document. | Robert Golden |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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