**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 | When accepting input, it is important to ensure that appropriate formatting is enforced based on the data being entered. Some examples are alphabet characters for names, digits for number inputs, and format like mm/dd/yyyy for dates. Character length should also be enforced based on the data input. |
| 1. Heed Compiler Warnings | Heeding compiler warnings will help the code avoid potential security vulnerabilities and can make the code more efficient. Using the highest level of warning detection will point out even smaller warnings that should be addressed, even if the code functions without addressing the warnings. |
| 1. Architect and Design for Security Policies | Application code must always abide by the company’s security policies. It should be designed in a way to allow easily updating users’ privileges whenever it is required. |
| 1. Keep It Simple | All code should be created to contain enough code to perform their functions with as little unnecessary extra code as possible. Code should be commented and contain a summary in the beginning of the code and short descriptions of each unique function. |
| 1. Default Deny | Deny all access to privileged information then authenticate based on permissions. No access should be granted by default. |
| 1. Adhere to the Principle of Least Privilege | Only grant permissions that are absolutely needed by the user. Review privileges frequently and change any that no longer require any special privileges. |
| 1. Sanitize Data Sent to Other Systems | Make sure that only the appropriate commands are sent through databases. Any unnecessary commands should be removed. |
| 1. Practice Defense in Depth | Implement more than one security measure that goes beyond just the secure coding. Running code in secure environments will help with any potential vulnerabilities that may not be addressed in the code. |
| 1. Use Effective Quality Assurance Techniques | All quality assurance processes should be properly followed and documented. Make sure that all processes are up to date and effective for the software being reviewed. |
| 1. Adopt a Secure Coding Standard | Coding security models should be consistent in the company. All developers must follow the company’s secure coding standards. |

## 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] | Data Type Coding Standard |

| **Noncompliant Code** |
| --- |
| Not following proper naming conventions for data types. Variables should be lowercase first character and uppercase character starting new word. |
| int Number;  int MyNumberVariable; |

| **Compliant Code** |
| --- |
| Variables should be lowercase first character and uppercase character starting new word. |
| int number;  int myNumberVariable; |

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

|  |
| --- |
| **Principles(s):** Every code developed must follow company standard. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| DCL53-CPP | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++2502, C++2510 |  |
| Klocwork | 2021.1 | CERT.DCL.AMBIGUOUS\_DECL |  |
| LDRA tool suite | 9.7.1 | 296 S | Partially implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-DCL53-a  CERT\_CPP-DCL53-b | Always declare functions at file scope  Identifier declared in a local or function prototype scope shall not hide an identifier declared in a global or namespace scope |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Data Value Coding Standard |

| **Noncompliant Code** |
| --- |
| Range check should occur before casting the enumerator type. The noncompliant code below shows the cast before the check occurs. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| Range check occurs before casting the enumerator type. |
| enum EnumType {    First,    Second,    Third  };    void f(int intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

|  |
| --- |
| **Principles(s):** Ensure range checks are always valid and run efficiently. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | assert\_failure |  |
| CodeSonar | 6.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TBA  LANG.MEM.TO  LANG.MEM.TU | Buffer overrun  Buffer underrun  Tainted buffer access  Type overrun  Type underrun |
| Helix QAC | 2021.2 | C++3162, C++3163, C++3164, C++3165 |  |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | String Correctness Coding Standard |

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

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

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

|  |
| --- |
| **Principles(s):** Keep It Simple, Adopt a Secure Coding Standard. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++4746, C++4747, C++4748, C++4749 |  |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |
|  |  |  |  |
|  |  |  |  |

### Coding Standard 4

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

| **Noncompliant Code** |
| --- |
| The customerName parameter passed through the request.getParameter() function is not validated, allowing any data entry to come in. |
| String query = "SELECT account\_balance FROM user\_data WHERE user\_name = "  + request.getParameter("customerName");  try {  Statement statement = connection.createStatement( ... );  ResultSet results = statement.executeQuery( query );  } |

| **Compliant Code** |
| --- |
| The Parameter string in CheckParameter() gets validated and will return false if a potential SQL injection attempt is found. |
| bool CheckParameter(string Parameter)  {  string key[14] = {"and","\*","="," ","%0a","%","/","union","|","&","^" ,"#","/\*","\*/" };  for (int i = 0; i < 14; i++)  {  if (Parameter.find(key[i]) != string::npos)  {  return false;  }  }  return true;  } |

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

|  |
| --- |
| **Principles(s):** Practice Defense in Depth and Validate Input Data. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++4075, C++4076 |  |
| LDRA tool suite | 9.7.1 | 527 S, 56 D, 71 D | Partially implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-ERR56-a | Ensure resources are freed |
| PRQA QA-C++ | 4.4 | 4075, 4076 |  |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Memory Protection Coding Standard |

| **Noncompliant Code** |
| --- |
| The code does not properly de-allocate memory. The pointer is passed to delete without being passed to new resulting in undefined behavior. |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    delete s1;  } |

| **Compliant Code** |
| --- |
| The code properly de-allocates memory by passing the pointer through new to be deleted. |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;    // ...    s1->~S();  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data and Architect and Design for Security Policies. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | invalid\_dynamic\_memory\_allocation  dangling\_pointer\_use | [Insert text.] |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM51 |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzer-unix.MismatchedDeallocator | Checked by clang-tidy, but does not catch all violations of this rule |
| CodeSonar | 6.1p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM | Free non-heap variable  Double free  Type mismatch |

### Coding Standard 6

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

| **Noncompliant Code** |
| --- |
| Explicitly state the purpose of the assertion if it will be utilized in the real code when the program is run. |
| #include <stdio.h>  #include <assert.h>    int main()  {  int x = 7;    x = 9;    assert(x==7);    /\* Rest of the code \*/    return 0;  } |

| **Compliant Code** |
| --- |
| Code comments state the purpose of the assertions. |
| #include <stdio.h>  #include <assert.h>    int main()  {  int x = 7;  // At some point in the program, the value may get unintentionally changed. The assignment below does that.    x = 9;    assert(x==7);    /\* Rest of the code \*/    return 0;  } |

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

|  |
| --- |
| **Principles(s):** Adopt a Secure Coding Standard and Keep It Simple. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | stdlib-use | Partially checked |
| CodeSonar | 6.1p0 | BADFUNC.ABORT | CodeSonar |
| Helix QAC | 2021.2 | C++5014 |  |
| Klocwork | 2021.1 | MISRA.CATCH.ALL |  |

### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| The throwing function below does not properly terminate and may result in another exception not properly handled. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  throwing\_func();  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

| **Compliant Code** |
| --- |
| The f() function properly handles the exceptions and terminates properly. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.  try {  throwing\_func();  } catch (...) {  // Handle error  }  }    int main() {  if (0 != std::atexit(f)) {  // Handle error  }  // ...  } |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | stdlib-use | Partially checked |
| CodeSonar | 6.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| Helix QAC | 2021.2 | C++5014 |  |
| Klocwork | 2021.1 | MISRA.CATCH.ALL  CERT.ERR.ABRUPT\_TERM |  |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Functions | [STD-008-CPP] | Functions Naming Convention Standard |

| **Noncompliant Code** |
| --- |
| Function naming convention does not follow the standard in examples below. |
| bool ExampleFunction();  int void another\_function(); |

| **Compliant Code** |
| --- |
| Function naming convention follows the standard. |
| bool exampleFunction();  int void anotherFunction(); |

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

|  |
| --- |
| **Principles(s):** Adopt a Secure Coding Standard and Keep It Simple. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| DCL53-CPP | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++2502, C++2510 |  |
| Klocwork | 2021.1 | CERT.DCL.AMBIGUOUS\_DECL |  |
| LDRA tool suite | 9.7.1 | 296 S | Partially implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-DCL53-a |  |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Comments | [STD-009-CPP] | Headers and Comments Standards |

| **Noncompliant Code** |
| --- |
| All code should contain a summary of the code at the top and name of the author, date of last edit, and version number if applicable. |
| /\*\*  \* Felix Vargas  \*  \*  \*/  #include <iostream>    using namespace std;    int main()  {  cout<<"Hello World";    return 0;  } |

| **Compliant Code** |
| --- |
| Code contains appropriate header information. |
| /\*\*  \* Simple program to print “Hello World”.  \* Author: Felix Vargas  \* Version: 1.0  \* Created: 7/18/2021  \*/  #include <iostream>    using namespace std;    // main function -  // where the execution of program begins  int main()  {  // prints hello world  cout<<"Hello World";    return 0;  } |

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

|  |
| --- |
| **Principles(s):** Adopt a Secure Coding Standard and Keep It Simple. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++2502, C++2510 |  |
| Klocwork | 2021.1 | CERT.DCL.AMBIGUOUS\_DECL |  |
| LDRA tool suite | 9.7.1 | 296 S | Partially implemented |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-DCL53-a  CERT\_CPP-DCL53-b | Always declare functions at file scope  Identifier declared in a local or function prototype scope shall not hide an identifier declared in a global or namespace scope |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Password | [STD-010-CPP] | Password Authentication Standard |

| **Noncompliant Code** |
| --- |
| The following code takes the string as an input but does not securely delete the code, making it potentially discoverable. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  if (!authenticate("correct code")) {  printf("Authentication error\n");  return -1;  }    printf("Authentication successful\n");  // ...Work with system...  return 0;  } |

| **Compliant Code** |
| --- |
| This code takes the input and uses it to validate the authentication and then deletes it securely to prevent any discovery of it. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  #define CODE\_LEN 50  char code[CODE\_LEN];  printf("Please enter your authentication code:\n");  fgets(code, sizeof(code), stdin);  int flag = authenticate(code);  memset\_s(code, 0, sizeof(code));  if (!flag) {  printf("Access denied\n");  return -1;  }  printf("Access granted\n");  // ...Work with system...  return 0;  } |

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

|  |
| --- |
| **Principles(s):** ValidateInput Data and Default Deny. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++4746, C++4747, C++4748, C++4749 |  |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |
|  |  |  |  |
|  |  |  |  |

## 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.

The diagram shows the process in which security protocols follow. In the pre-production phase, breach or attach simulations help prepare an appropriate response based on the attack and severity. During production, we should do our best to follow proper damage mitigation protocols when an attack takes place.

## 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 |
| DCL53-CPP | Medium | Unlikely | Medium | Medium | 3 |
| STD-003-CPP | High | Probable | High | High | 2 |
| STD-004-CPP | High | Likely | High | High | 2 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Probable | Medium | Low | 3 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | Medium | Medium | Medium | Low | 3 |
| STD-009-CPP | Low | Unlikely | Medium | Low | 3 |
| STD-010-CPP | High | Probable | High | 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 | Data stored in a data center should be encrypted and only accessible via appropriate methods to ensure it is decrypted. This prevents unauthorized visibility to the data should there be a hack. |
| Encryption at flight | Data being sent over the internet should be encrypted before it leaves the internal network. This helps prevent man in the middle attacks. |
| Encryption in use | Confidential data is to be accessed only by those who are authorized. Drives containing that data should be encrypted when their use is done. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Make sure that the user’s credentials match what is on file. |
| Authorization | Provide access based on the level of access the user has. |
| Accounting | Log every log in and user activity. |

**\***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 | 08/08/2020 | Final draft | Felix Vargas | [Insert text.] |
| [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 |