**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 working with any sort of coded program carefully make sure what you are inputting is what needs inputting. If you, do not it could cause for issues with program and lead to people having to try and fix anything that is broken. Its better to take your time then rush through. |
| 1. Heed Compiler Warnings | If when compiling code, you get any warnings take your time and fix them. Do not just ignore them because its either going to do two things break the entire program and make it so you or someone else needs to go and fix it later. Or its just not going to work and you might break something else. |
| 1. Architect and Design for Security Policies | This is important to take time and figure out how all the safety precautions are set up to make sure there are no issues that will cause harm. This should be done for all systems in any sort of network. |
| 1. Keep It Simple | When it comes to security having more complex ideas can cause issues between programs and other users. You might think having the most complex solution would be best no matter what. But the best way to go about it is figure out what is needed and only use that. |
| 1. Default Deny | This is important because you should have a list of programs and other items that should be allowed. Then if someone tries to use anything else it should get denied because it might be harmful. If it is something that is really needed the correct person should make the exception that it can be used. |
| 1. Adhere to the Principle of Least Privilege | Each system, person, etc. should only have the privilege that is needed to work. Doing this allows for there to be less ways for someone to take advantage of one of those things and do harm with it. |
| 1. Sanitize Data Sent to Other Systems | This is to remove anything from data that doesn’t need to be there before its sent. Also if its sent to another system and no longer needs to be on the system it was sent from it should be removed. |
| 1. Practice Defense in Depth | You should no count on only one defense item to do everything you need. You should have layers of different ones to create a safety net for what might be going on. You should also look at what the needs of the business are to figure out how far to go in depth. |
| 1. Use Effective Quality Assurance Techniques | This allows for vulnerabilities to be figured out while still developing whatever is being developed. This means you should constantly be testing what ever is in development and have people whose main job is to find weakness and issues. This will make the final product better. |
| 1. Adopt a Secure Coding Standard | Having a securing coding standard means for everyone who might be working with you on a project or working on something similar will understand what they are looking at. This also means that if you go from working with one group of people on developing something then need to go help someone else you can just jump in and understand what needs to be done. If not, you will need to figure out what standard they are using and try to learn it if you do not understand it. |

## 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-C++] | Implement abstract data types using opaque types |

| **Noncompliant Code** |
| --- |
| The managed string and functions are defined in the string\_m.h file. With the inclusion of string\_mx it allows for programmers to mess with the code easier. |
| struct string\_mx {  **size\_t** size;  **size\_t** maxsize;    unsigned **char** strtype;  **char** \*cstr;  };    typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| The correcting thing to do is make the string\_mx as a private type making it harder to alter where it shouldn’t be messed with. |
| struct string\_mx;  typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/ |

**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 | Low | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-C++] | Value-returning functions must return a value from all exit paths |

| **Noncompliant Code** |
| --- |
| The input is not returned so nothing is shown when running the code. |
| **int** absolute\_value(**int** a) {    if (a < 0) {      return -a;    }  } |

| **Compliant Code** |
| --- |
| By adding the return a; part the code will actually return the absolute value. |
| **int** absolute\_value(**int** a) {    if (a < 0) {      return -a;    }    return a;  } |

**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 | Likely | Medium | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| This code is unbounded so it is going to cause a buffer overflow and make it easier for people to cause harm to the code. |
| #include <iostream>    void f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Using std::string allows for protection against |
| #include <iostream>  #include <string>    void f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

**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 | Medium | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C++] | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;    class Login {    public Connection getConnection() throws SQLException {      DriverManager.registerDriver(new              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"      return DriverManager.getConnection(dbConnection);    }      String hashPassword(char[] password) {      // Create hash of password    }      public void doPrivilegedAction(String username, char[] password)                                   throws SQLException {      Connection connection = getConnection();      if (connection == null) {        // Handle error      }      try {        String pwd = hashPassword(password);          String sqlString = "SELECT \* FROM db\_user WHERE username = '"                           + username +                           "' AND password = '" + pwd + "'";        Statement stmt = connection.createStatement();        ResultSet rs = stmt.executeQuery(sqlString);          if (!rs.next()) {          throw new SecurityException(            "User name or password incorrect"          );        }          // Authenticated; proceed      } finally {        try {          connection.close();        } catch (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| public void doPrivilegedAction(    String username, char[] password  ) throws SQLException {    Connection connection = getConnection();    if (connection == null) {      // Handle error    }    try {      String pwd = hashPassword(password);        // Validate username length      if (username.length() > 8) {        // Handle error      }        String sqlString =        "select \* from db\_user where username=? and password=?";      PreparedStatement stmt = connection.prepareStatement(sqlString);      stmt.setString(1, username);      stmt.setString(2, pwd);      ResultSet rs = stmt.executeQuery();      if (!rs.next()) {        throw new SecurityException("User name or password incorrect");      }        // Authenticated; proceed    } finally {      try {        connection.close();      } catch (SQLException x) {        // Forward to handler      }    }  } |

**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 | High | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C++] | Do not store an already-owned pointer value in an unrelated smart pointer |

| **Noncompliant Code** |
| --- |
| P1 and P2 when deleted end up deleting the same pointer value. |
| void f() {  **int** \*i = new **int**;    std::shared\_ptr<**int**> p1(i);    std::shared\_ptr<**int**> p2(i);  } |

| **Compliant Code** |
| --- |
| This makes when it happens that P1 goes to zero instead of deleting completely. |
| 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):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C++] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| This code ends up having std:basic\_filebuf<T> and what it controls not closing properly after it is used. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| By using std::terminate this has everything properly close after use. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    file.close();    if (file.fail()) {      // Handle error    }    std::terminate();  } |

**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 | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-C++] | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| This code block ends up stop because throwing\_func() can end up causing an error. |
| #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** |
| --- |
| Changing it to make f() to handle all exceptions so they don’t get thrown again which fixes the error. |
| #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):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programing | [STD-008-C++] | Do not use pointer-to-member operators to access nonexistent members |

| **Noncompliant Code** |
| --- |
| D::g ends up getting called on instead of B where it needs to be called on. This results in undefined behavior and causes issues. |
| struct B {    virtual ~B() = default;  };    struct D : B {    virtual ~D() = default;    virtual void g() { /\* ... \*/ }  };    void f() {    B \*b = new B;      // ...      void (B::\*gptr)() = static\_cast<void(B::\*)()>(&D::g);    (b->\*gptr)();    delete b;  } |

| **Compliant Code** |
| --- |
| The solution is to make it so B::g shown to actually work and not mess up like up top. |
| struct B {    virtual ~B() = default;  };    struct D : B {    virtual ~D() = default;    virtual void g() { /\* ... \*/ }  };    void f() {    B \*b = new D; // Corrected the dynamic object type.      // ...    void (D::\*gptr)() = &D::g; // Moved static\_cast to the next line.    (static\_cast<D \*>(b)->\*gptr)();    delete b;  } |

**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 | Likely | Medium | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-009-C++] | Do not alternately input and output from a file stream without an intervening positioning call |

| **Noncompliant Code** |
| --- |
| The beahvior is undefined because there is nothing showing position between output and input. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";    std::string str;    file >> str;  } |

| **Compliant Code** |
| --- |
| This is fixed here because both input and output gets called on separately. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";      std::string str;    file.seekg(0, std::ios::beg);    file >> str;  } |

**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 | Likely | Medium | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers | [STD-010-C++] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| The enum value does not get checked before the conversion and get end up with a unspecified value being displayed. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This checks the values from the enum before anything happens to make sure the conversion works properly. |
| 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):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CPP | 2.4.1 | CPPcheck | Cppcheck is a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. |

## 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.

Automation allows for you to have an easier time once you get all policies set up. So, at first you usually need to create solutions to the policies. Then after that you can automate them so if they get detected you can fix them without having to go back. CPPcheck has started doing that because it tells you what is wrong with each portion of code and what needs to be changed. Then you can go to each line of code and make the change and be good to go.

## 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-C++ | High | Unlikely | Low | High | 2 |
| STD-002-C++ | Medium | Likely | Medium | Low | 1 |
| STD-003-C++ | High | Unlikely | Medium | High | 3 |
| STD-004-C++ | High | Likely | High | High | 3 |
| STD-005-C++ | Medium | Medium | Medium | Medium | 2 |
| STD-006-C++ | Low | Unlikely | Low | Low | 1 |
| STD-007-C++ | Medium | Unlikely | Medium | High | 2 |
| STD-008-C++ | Medium | Likely | Medium | Low | 2 |
| STD-009-C++ | Low | Likely | Medium | Low | 1 |
| STD-010-C++ | Low | Unlikely | Low | Low | 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 | Data being stored needs to be encrypted so it can not be accessed by anyone who should not have access to it. |
| Encryption at flight | When transferring data that is sensitive or considered classified, it should be encrypted to stop anyone who should gain access to having access. |
| Encryption in use | When accessing data, you should have it encrypted, just so no one can see what you are working on that shouldn’t see it. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication allows for a check to be made top make sure the person who is logging in, is who they are supposed to be. |
| Authorization | Authorization allows for a check to be made for whoever is logging in, so they have access to the right information they should have. |
| Accounting | Accounting means keeping track of what everyone is doing when accessing the software from the company. This just keeps everyone accountable so if they do something wrong, it can be delt with fast. |

**\***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 | 6/13/2021 | Final | Connor Blauch |  |
| [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 |