**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 it comes from untrusted sources this helps validates them. Software vulnerabilities can be prevented using validating input data. Command line arguments, environmental variables and even network interfaces when it comes to these you should always be aware. |
| 1. Heed Compiler Warnings | After compiling the code pay attention to the warning levels and make sure to correct the errors before continuing. The warnings can show the developer what line to look in order to correct it. This can detect lines of code that you made an error with. |
| 1. Architect and Design for Security Policies | Security policies should be implemented when you are in the design process of the software, that is how serious security policies should be taken. |
| 1. Keep It Simple | Complex systems can take a while to get used too when it comes to putting them together but if you learn simplify your code you will be keeping it simple. |
| 1. Default Deny | If you set up decisions based on permissions other than exclusive it will become the standard for you. |
| 1. Adhere to the Principle of Least Privilege | To help reduce the chances of attackers you must adhere to the principle of least privilege and this shows how important this can be, they should only be accessed using the appropriate time it takes. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data is used to steer away attackers that try to manipulate the components of subsystems. |
| 1. Practice Defense in Depth | Practicing creating multiple layers of security will prove that you have practiced defense in depth. Several layers can fail but there will be other layers that can handle the defense. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques can help prevent future maintenance costs because each step can be signed off and checked so that every inch of progression can be verified as complete. |
| 1. Adopt a Secure Coding Standard | Practice makes perfect and I think if you really adapt a secure coding standard then you can really code securely like it is second nature. This should be applied to all languages that you write. |

### 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] | Enumeration value that is out of range should not be casted. |

| **Noncompliant Code** |
| --- |
| To see if a given value is within a given range of acceptable enumeration values us this noncompliant code. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| In order to have compliant code, the code needs to check the value before handling the conversion to prove that it doesn’t have a unspecified value. |
| 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):** It is highly possible for values to have a buffer flow. I have learned that enumerators are rarely used for indexing. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.90 | CertC++ - INT50 |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ | 2020.2 | CERT\_CPP – INT50-a | Expression with Enum underlying values |
| PRQA QA-C++ | 4.4 | 3013 |  |
| PVS -Studio | 7.07 | V1016 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | References that are valid should be used, pointers and also iterators of the container. |

| **Noncompliant Code** |
| --- |
| Pos is invalidated after first call. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| Pos is assigned to a valid iterator. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  pos = d.insert(pos, items[i] + 41.0);  }  } |

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

| **Principles(s):** References that are invalid, pointers, or iterators in regards to undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Overflow\_unpon\_dereference |  |
| Helix QAC | 2021  .1 |  |  |
| Parasoft C/C++ | 2020.2 | CERT\_CPP-CTR51-A | Do not modify Container |
| PVS- Studio | 7.07 | V783 |  |

**Coding Standard 3003**

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | If a std::string is from a null pointer do not attempt |

| **Noncompliant Code** |
| --- |
| A string has been created from the results of std::getenv(). |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| Getenv() must be checked before the string. |
| #include <cstdlib>  #include <string>    void f() {  const char \*tmpPtrVal = std::getenv("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** Null pointers is undefined behavior, this could cause execution of arbitrary code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Assert\_failure |  |
| Helix QAC | 2021.1 |  |  |
| ParasoftC/C++ | 2020.2 | CERT\_cpp- STR51-a | Avoid null Pointer dereferencing |

#### 

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | If the pointer value is already owned do not store in smart pointer. |

| **Noncompliant Code** |
| --- |
| Two smart pointers that are unrelated are connected to the same pointer value. |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| #include <memory>  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** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Dangling\_pointer\_use |  |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ Test | 2020.2 | CERT\_CPP-MEM56-A | If the pointer value is already owned do not store in smart pointer. |
| Pvs – Studio | 7.01 | V1006 |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Allocated resources should deallocate. |

| **Noncompliant Code** |
| --- |
| The new operator is placed and passed the expression to the placement.it also attempts to free data that was not returned to the new operator. |
| #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** |
| --- |
| This code removes the function operator delete. |
| #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):** Vulnerabilities can be caused not matching allocation function results, |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Invalid\_dymanic\_memory\_allocation\_dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 6.9.0 | VERTC++-MEM51 |  |
| Clang | 3.9 | clang-analyzercplusplus.NewDeleteLeaks  -Wmismatched-new-delete  clang-analyzerunix.MismatchedDeallocator | Checked by clang-tidy, but  does not catch all violations of  this rule |
| CodeSonar | 6.0p0 | ALLOC.FNH  ALLOC.DF  ALLOC.TM | DOUBLE FREE  TYPE MISMATCH |
| Helix QAC | 2021.1 |  |  |
| Klocwork | 2021.1 | CL.FFM.ASSIGNFM  CL.FFM.COPY  CL.FMM  FMM.MIGHT  FMM.MUST  FNH.MIGHT  FNH.MUST  FUM.GEN.MIGHT  FUM.GEN.MUST  UNINIT.CTOR.MIGHT |  |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470  S, 483 S, 484 S, 485 S, 64 D, 112 D | Partially implemented |
| Parasoft C/C++  test | 2020.2 | CERT\_CPP-MEM51-a  CERT\_CPP-MEM51-b  CERT\_CPP-MEM51-c  CERT\_CPP-MEM51-d | Properly deallocate dynamically  allocated resources |
| Parasoft Insure ++ |  |  | Runtime Detection |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Static assertion is used to test the value of the constant expression |

| **Noncompliant Code** |
| --- |
| A memory map structure which has a purpose for code to behave |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned  int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| This code represents a compliant solution with a preprocessor conditional statement. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned  int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** Just because there is an absence of static assertions does not validate the code to being correct. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CERTC-DCL03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.90p0 |  | Checking reports using assert() macro |
| Compass/Rose | [Insert text.] |  | NO violations of the rule |
| Éclair | 1.2 | CC2.DCL03 | Fully Implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Begins executing before main() in regards to all exceptions |

| **Noncompliant Code** |
| --- |
| Not caught when globalS is executed during the first run of the program |
| struct S {  S() noexcept(false);  };  static S globalS; |

| **Compliant Code** |
| --- |
| No need to modify the source code, because it is replace by a function call. |
| struct S {  S() noexcept(false);  };    S &globalS() {  try {  static S s;  return s;  } catch (...) {  // Handle error, perhaps by logging it and gracefully terminating the application.  }  // Unreachable.  } |

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

| **Principles(s):** Program termination is because there is an exception that cant be found. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | potentially-throwing static-initialization | Partially checked |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++-err58 |  |
| Clang | 3.9 | Cert-eer58-cpp | Checked by Clang Tidy |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++  test | 2020.2  CERT\_CPPERR58-a |  |  |
| PRQA QA-C++ | 4.4 | 4634, 4636, 4637, 4639 | Partially Checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-008-CPP] | Output and input should not be placed from a file stream |

| **Noncompliant Code** |
| --- |
| Reads from the same file |
| #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** |
| --- |
| Bewtten input and out std::basic\_istream<T>::seekg() function is summoned |
| #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):** if you do not intervene the flush it will result in undefined behavior |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.1 |  |  |
| Parasoft C/C++ Test | 2020.2 | CERT\_CPP\_FIO50-a | Positioning call |
| Polyspace Bug Finder | R2020a | ECRT C++: FIO50-CPP | Searches for input and output |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-010-LLL] | Virtual functions should not be messed with that are connected to constructors or destructors. |

| **Noncompliant Code** |
| --- |
| Each functions relates to the other. It throws the program off by using virtual functions |
| struct B {  B() { seize(); }  virtual ~B() { release(); }    protected:  virtual void seize();  virtual void release();  };  struct D : B {  virtual ~D() = default;    protected:  void seize() override {  B::seize();  // Get derived resources...  }    void release() override {  // Release derived resources...  B::release();  }  }; |

| **Compliant Code** |
| --- |
| These are non virtual. |
| Class B {  void seize\_mine();  void release\_mine();    public:  B() { seize\_mine(); }  virtual ~B() { release\_mine(); }  protected:  virtual void seize() { seize\_mine(); }  virtual void release() { release\_mine(); }  };  class D : public B {  void seize\_mine();  void release\_mine();    public:  D() { seize\_mine(); }  virtual ~D() { release\_mine(); }  protected:  void seize() override {  B::seize();  seize\_mine();  }    void release() override {  release\_mine();  B::release();  }  }; |

**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 | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | virtual-call-in-constructor  invalid\_function\_pointer | Fully Checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-OOP50 |  |

### 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.

[Insert your written explanations here.]

The first step would be to determine if there should be changes made to the authorization levels for the employee. Auditing is a vital step in this process for the company. Then you will need to properly set up authentication regarding user log ins to make sure everyone has the correct access levels that they need.

Testing will be incorporated into the plans for the development of the software. This should be applied to utilize mitigation tools this will prevent the code from not being secure enough to handle people’s personal data. Once all this information in the system and users can log in, it will need to be monitored consistently in order to make sure things are running smoothly.

### 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-CLG | Low | Unlikely | High | P1 | L3 |
| STD-002-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-003-CLG | High | Likely | Low | P27 | L1 |
| STD-004-CLG | Low | Likely | Low | P9 | L2 |
| STD-005-CLG | Medium | Probable | Medium | P8 | L2 |
| STD-006-CLG | Medium | Unlikely | Medium | P4 | L3 |
| STD-007-CPP | High | Likely | High | P9 | L2 |
| STD-008-CPP | Low | Likely | Medium | P6 | L2 |
| STD-009-CPP | High | Likely | Medium | P18 | L1 |
| STD-010-CPP | Low | Probable | High | P2 | L3 |

### 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 | Converting one data to another happens in the encryption in rest process. The algorithm that is used in this step is only authorized by a user who is qualified. Sensitive data is stored and must be secure properly on the severs and systems. This helps from breaches and attacks. |
| Encryption at flight | Encrypting data that is being transmitted is important to protect data. Data will not only be transferred internally but externally as well, and that needs to be monitored as well, so that there is no loop holes while sending data. This makes data vulnerable letting users send data so this encryption process must be completed before sending. |
| Encryption in use | This means to never leave data in an unsecure state. The data should always be encrypted regardless of the location and source. Data that is encrypted will stay secure and harder for hackers to attack the server. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This method is to make sure the users have access to the data that they are qualified to access in the server. A unique set of credentials are needed to access certain information. |
| Authorization | Profiles need permissions to change data, access information, and make system changes. Users only need to be able to access the section in which they need to do their job. |
| Accounting | This is where the system and data should be monitored to ensure everything is operating smoothly. The levels of access is also monitored in this process and should be verified consistently to make sure each users has the correct level of access. |

**\***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 | 6/19/2022 | Risk established | Dustin Jackson | Trevor Hodde |
| 2.1 | 6/26/2022 | Risk Assessments | Dustin Jackson | Trevor Hodde |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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