**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 | All untrusted data source inputs should be validated. Highest concern items are external data sources such as command line arguments, network interfaces, environmental variables, and user controlled files. |
| 1. Heed Compiler Warnings | Use the highest warning level available for the compiler being used. Use static and dynamic testing to detect and eliminate security flaws. |
| 1. Architect and Design for Security Policies | Software should be designed to implement and enforce security policies. If different privileges are required at different times, divide the code into separate subsystems. |
| 1. Keep It Simple | Always attempt to maintain simple and minute code. Overcomplexity of code can lead to house errors and open up potential security breaches. |
| 1. Default Deny | Always code for a denial of access at default and focus on permission over exclusion. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. |
| 1. Sanitize Data Sent to Other Systems | Data must be sanitized before being passed to other subsystems in order to avoid SQL, command or other injection attacks. |
| 1. Practice Defense in Depth | Approach risk with multiple defensive strategies. Create a system of redundancy in order to protect the operation if one layer of defense shall fail. |
| 1. Use Effective Quality Assurance Techniques | Using assurance techniques such as fuzz testing, penetration testing, and source code audits are effective in exposing vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard. |

### 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] | All functions, return types, and variables must be declared before using them. |

| **Noncompliant Code** |
| --- |
| Unspecified (implicit) variable type |
| foo = 0; |

| **Compliant Code** |
| --- |
| Specified variable type |
| int foo = 0; |

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

| **Principles(s):**  1: Defined data types should match input.  2: Correct all compiler warnings.  4: Practice explicit variable declarations.  8: Add a layer of defense by using variable, function, and return declarations. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’s | 20.10 | Type-Specifier  Function-return-type  Implicit-function-declaration  Undeclared-parameter | Fully Checked |
| ÉCLAIR | 1.2 | [CERTC-DCL31](https://wiki.sei.cmu.edu/confluence/display/c/DCL31-C.+Declare+identifiers+before+using+them) | Fully Implemented |
| Axivion Bauhaus Suite | 7.2.0 | CERTC-DCL31 | Fully Implemented |
| Coverity | 2017.07 | MISRA C 2012 Rule 8.1 | Implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-001-CPP] | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| Uninitialized local variable is evaluated as part of print expression, resulting in undefined behavior. |
| #include <iostream>    void f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| Local variable is initialized prior to printing. |
| #include <iostream>    void f() {  **int** i = 0;    std::cout << i;  } |

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

| **Principles(s):**  1: Input shall be within ranged defined by variable.  2: Correct all compiler warnings.  8: Add a layer of defense by initializing all variables. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Uninitialized – read | Partially Checked |
| LDRA tool Suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially Implemented |
| Parasoft C/C++ test | 2021.2 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| RuleChecker | 20.10 | Uninitialized - read | Partially Checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Storage must has enough space for null terminator. |

| **Noncompliant Code** |
| --- |
| Code unbounded, resulting in buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| With std, buffer overflow is prevented. |
| #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):**  1: Input string storage must allow null terminator.  7: SQL injection and buffer overflow can be prevented by string lengths that are bounded by null terminators.  8: Provide a layer of defense by verifying string length. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| Helix QAC | 2022.1 | **C++2835, C++2836, C++2839, C++5216** | N/A |
| LDRA tool suite | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially Implemented |
| Parasoft C/C++ test | 2021.2 | **CERT\_CPP-STR50-b** **CERT\_CPP-STR50-c** **CERT\_CPP-STR50-e** **CERT\_CPP-STR50-f** **CERT\_CPP-STR50-g** | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004--C] | Maintain use of prepared statements. |

| **Noncompliant Code** |
| --- |
| No prepared statement, risk of SQL injection. |
| uName = getRequestString("username");  uPass = getRequestString("userpassword");  sql = “SELECT \* FROM Users WHERE Name = " + uName + " AND Pass = " +  uPass + ” |

| **Compliant Code** |
| --- |
| Prepared statement. Injection potential limited. |
| PreparedStatement pStmt = PreparedStatement();  std::cin >> username;  std::cin >> userpassword;  sql = “SELECT \* FROM Users WHERE Name = %s AND Pass = %s;”, username,  userpassword}; |

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

| **Principles(s):**  1: Validate input to prevent SQL injection  3: Use prepared SQL statements.  7: Usage of prepared statements sanitizes data to reduce chances of SQL injection. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ParaSoft C/C++ test | 2020.2 | F1O30-C | Don’t use unfiltered data |
| Astree’s | 20.10 | N/A | Taint analysis/Stubbing |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| Variable S is dereferenced after it has been deleted resulting in a write-after-free vulnerability. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| S is removed after it is no longer useful. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

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

| **Principles(s):**  2: Use tools to identify, detect, and mitigate freed memory issues.  9: Implement QA standards to identify freed memory occurrences. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree’s | 20.10 | Dangling\_pointer\_use | N/A |
| Axivion Bauhaus suite | 7.2.0 | CertC++ -MEM50 | N/A |
| LDRA tool suite | 9.7.1 | 483 S, 484 S | Partially Implemented |
| Parasoft C/C++ test | 2021.2 | CERT\_CPP-MEM50-a | Do not use resourced that have been freed. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertions should be used to verify the status of a pointer. Use of an abort function is used to prevent errors. |

| **Noncompliant Code** |
| --- |
| Pointed may be null, causing an issue with functionality. |
| /\* assert example \*/  #include <stdio.h> /\* printf \*/  #include <assert.h> /\* assert \*/  void print\_number(int\* myInt) {  printf ("%d\n",\*myInt);  }  int main ()  {  int a = 10;  int \* b = NULL;  int \* c = NULL;  b=&a;  print\_number (b);  print\_number (c);  return 0;  } |

| **Compliant Code** |
| --- |
| Assert used to prevent erorrs. |
| /\* assert example \*/  #include <stdio.h> /\* printf \*/  #include <assert.h> /\* assert \*/  void print\_number(int\* myInt) {  assert (myInt!=NULL);  printf ("%d\n",\*myInt); |

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

| **Principles(s):**  9: Use assertions to test code through the program’s entirety. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2020.2 | ERR56-CPP | Partially Implemented |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| No handlers for exceptions. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| Handler inserted. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

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

| **Principles(s):**  3: Use of try/catch/throw tools to prevent incorrect stoppage of a program.  9: Implemented continuous test throughout code and throw exceptions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2020.2 | ERR51-CPP | Always catch exceptions. |
| Astree’s | 20.10 | Catch\_all  Catch\_early | Partially Checked |
| Polyspace Bug Finder | R2020a | ERR51-CPP | Checks unhandled exceptions |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Correct integer precisions | [STD-008--C] | Use correct integer precisions. |

**Location: https://wiki.sei.cmu.edu/confluence/display/c/INT35-C.+Use+correct+integer+precisions**

| **Noncompliant Code** |
| --- |
| Function can result in values for “exp” that are too large. |
| #include <limits.h>    unsigned int pow2(unsigned int exp) {    if (exp >= sizeof(unsigned int) \* CHAR\_BIT) {      /\* Handle error \*/    }    return 1 << exp;  } |

| **Compliant Code** |
| --- |
| Uses the “popcount” function to determine the precision of any integer type. |
| #include <stddef.h>  #include <stdint.h>    /\* Returns the number of set bits \*/  size\_t popcount(uintmax\_t num) {    size\_t precision = 0;    while (num != 0) {      if (num % 2 == 1) {        precision++;      }      num >>= 1;    }    return precision;  }  #define PRECISION(umax\_value) popcount(umax\_value) |

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

| **Principles(s):**  1: Ensuring precision of input data.  9: Utilize QA standards to maintain precise data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | N/A | Supported: Astree reports overflows due to insufficient precision. |
| Helix QAC | 2022.1 | C0582  C++3115 | N/A |
| Parasoft C/C++ test | 2021.2 | CERT\_C-INT35a | Use correct integer precisions when checking the right hand operand of the shift operator |
| Polyspace Bug Finder | R2021a | CERT C: Rule INT35-C | Checks for situations when integer precisions are exceeded (rule fully covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Dereferencing Null Pointers | [STD-009--C] | Do not dereference null pointers. |

**Location: https://wiki.sei.cmu.edu/confluence/display/c/EXP34-C.+Do+not+dereference+null+pointers**

| **Noncompliant Code** |
| --- |
| Null pointer dereferenced. |
| #include <png.h> /\* From libpng \*/  #include <string.h>    void func(png\_structp png\_ptr, int length, const void \*user\_data) {    png\_charp chunkdata;    chunkdata = (png\_charp)png\_malloc(png\_ptr, length + 1);    /\* ... \*/    memcpy(chunkdata, user\_data, length);    /\* ... \*/   } |

| **Compliant Code** |
| --- |
| Ensures pointer returned by “png\_malloc()” is not null. |
| #include <png.h> /\* From libpng \*/  #include <string.h>     void func(png\_structp png\_ptr, size\_t length, const void \*user\_data) {    png\_charp chunkdata;    if (length == SIZE\_MAX) {      /\* Handle error \*/    }    if (NULL == user\_data) {      /\* Handle error \*/    }    chunkdata = (png\_charp)png\_malloc(png\_ptr, length + 1);    if (NULL == chunkdata) {      /\* Handle error \*/    }    /\* ... \*/    memcpy(chunkdata, user\_data, length);    /\* ... \*/     } |

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

| **Principles(s):**  2: Anticipate compiler warnings.  8: Establish multiple layers to avoid errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Null-dereferencing | Fully Checked |
| CodeSonar | 6.2p0 | **LANG.MEM.NPD LANG.STRUCT.NTAD LANG.STRUCT.UPD** | Null pointer dereference Null test after dereference Unchecked parameter dereference |
| ParasoftC/C++ test | 2021.2 | CERT\_C-EXP34-a | Avoid null pointer dereferencing |
| PRQA QA-C | 9.7 | 2810, 2811, 2812, 2813 | Fully Implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-010-CPP] | Use valid iterator ranges |

| **Noncompliant Code** |
| --- |
| Incorrect values are passed to the for\_each() function. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.end(), c.begin(), [](int i) { std::cout << i; });  } |

| **Compliant Code** |
| --- |
| Two iterators are passed correctly to the for\_each() function. |
| #include <algorithm>  #include <iostream>  #include <vector>    void f(const std::vector<int> &c) {  std::for\_each(c.begin(), c.end(), [](int i) { std::cout << i; });  } |

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

| **Principles(s):**  3: Iterators are protected from overflow.  4: Simple code clearly defines iterator ranges.  8: Adds a layer of defense by preventing iterator overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Overflow\_upon\_dereference | N/A |
| Parasoft C/C++ test | 2020.2 | CTR53a/b-CPP | Avoid incorrect iterator range. Avoid comparison of iterators from different containers. |

### 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.]

### 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 | Low | Unlikely | Low | 3 | 3 |
| STD-001-CPP | High | Probably | Medium | 12 | 1 |
| STD-003-CPP | High | Likely | Medium | 18 | 1 |
| STD-004—C | High | Likely | Medium | 18 | 1 |
| STD-005-CPP | High | Likely | Medium | 18 | 1 |
| STD-006-CPP | High | Likely | High | 9 | 2 |
| STD-007-CPP | Low | Probable | Medium | 4 | 3 |
| STD-008—C | Low | Likely | Medium | 2 | 3 |
| STD-009—C | High | Likely | Medium | 18 | 1 |
| STD-010-CPP | High | Probable | High | 6 | 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 | This encryption secures data where it is stored i.e. on a hard drive inside of a computer or mobile device, in a cloud storage program, or in a database. |
| Encryption at flight | This encryption protects data while it is being transferred from one storage location to another. For example, when sending emails or when submitting attachments to secure websites. |
| Encryption in use | This encryption protects data while it being edited, processed, viewed, created, or accessed. It is important to encrypt data in use because previous memory can be accessed by a breach and encryption keys can be exposed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying a user’s credentials. At the most basic layer it can be a simple user ID and password combination. Recently, most websites and other programs require utilization of the “2 step authentication” which adds another layer of security via biometrics or text/email codes. |
| Authorization | Authorization is the establishment of permissions for different users of a website or program. This controls the level of access to certain files or directories. For example, a base level employee will be able to login to a time clock website for their workplace, but only their manager (with a raised level of authorization compared to the base level employee) will be able to edit and approve the login history of that employee. |
| Accounting | Accounting is the establishment of a historical log of activity. One example of accounting is an “audit trail”. The audit trail tracks any activity that takes place in a website or program and that information cannot be altered or changed by anyone. This information can later be analyzed by individuals in an investigation or company audit. |

**\***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

**\*\*This is completed above each coding standard.\*\***

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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [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 |