**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 | We should validate all inputs to prevent vulnerabilities. We must consider eternal data sources such as network interfaces, and command line arguments to be unsafe. |
| 1. Heed Compiler Warnings | Security discrepancies that are found through testing need to be dealt with by editing the code to rid of compiler warnings. |
| 1. Architect and Design for Security Policies | When we design/build software we need to adhere to strict security policies. |
| 1. Keep It Simple | The saying “keep it simple stupid” can apply to coding as well, keeping it simple can sometimes negate errors and over complication. |
| 1. Default Deny | User credentials need to be set up with permissions so unwanted users can’t access system information. |
| 1. Adhere to the Principle of Least Privilege | We need to ensure that functions that are being performed are being executed with a little amount privilege like users and processes. |
| 1. Sanitize Data Sent to Other Systems | Making sure we remove things like SQL injection and buffer overflows before we pass it to another system. |
| 1. Practice Defense in Depth | We must ensure we protect systems with layers. Using multiple strategies is the best way to prevent vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | A good way to ensure effective quality assurance techniques could be to involve a third- party company that can come in and review code for potential vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Coding standards must be adhered to for each language because potential problems come with each different language. |

### 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] | Prior to using variables or functions, they must be declared. |

| **Noncompliant Code** |
| --- |
| Implicit VAR type |
| foo = 0; |
| #include <limits.h>  #include <stdio.h>    foo(void) {  return UINT\_MAX;  }    int main(void) {  long long int c = foo();  printf("%lld\n", c);  return 0;  } |

| **Compliant Code** |
| --- |
| Explicit VAR type |
| int foo = 0; |
| #include <limits.h>  #include <stdio.h>    **unsigned int** foo(void) {  return UINT\_MAX;  }  int main(void) {  long long int c = foo();  printf("%lld\n", c);  return 0; } |

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

| **Principles(s):**  1) Matching defined data types  2) Correcting ALL complier warning that have to do with the data types  3) When we add return declarations we add a layer of strength |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | NOT LIKELY | LOW | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astre’e | 22.10 | Checks type-specifier | Full checks |
| ECLAIR | 3.12.0 | [CERTC-DCL31](https://wiki.sei.cmu.edu/confluence/display/c/DCL31-C.+Declare+identifiers+before+using+them) | Full implementation |
| Parasoft | 2022.2 | [CERTC-DCL31](https://wiki.sei.cmu.edu/confluence/display/c/DCL31-C.+Declare+identifiers+before+using+them) | Declarations of all functions |
|  |  |  |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Uninitialized VAR should be avoided. |

| **Noncompliant Code** |
| --- |
| A local VAR that is uninitialized has the potential to output strange behaviors. |
| include <iostream>    void f() {    int i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| Initializing the local VAR prior to the print statement. |
| #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) Staying within the defined range  2) Correcting ALL complier warnings that have to do with uninitialized VARS  3) Initializing adds layers of strength |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | MOST LIKELY | MEDIUM | P12 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astre’e | 22.10 | uninitialized | Partial check |
| Parasoft | 2022.2 | [EXP53-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory) | Not before initialization |
| Polyspace | R2022b | [EXP53-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory) | Uninitialized VAR |
|  |  |  |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Ensuring the string has enough storage assures space for the null terminator. |

| **Noncompliant Code** |
| --- |
| An input without limit can cause buffer overflow issues. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Instead we can use std::string to prevent the buffer overflow for occurring. |
| #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) Allowing for NULL terminatory space  2) When strings are bound by NULL terminator length it prevents SQL injections and possible buffer overflows  3) Verifying string length |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | MOST LIKELY | MEDIUM | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 5.4 | [STR50-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator) | Not leaving space for NULL terminator |
| Parasoft | 2022.2 | [STR50-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator) | No overflows when writing to a buffer |
| Polyspace | R2022b | [STR50-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator) | Buffer overflow  No NULL in array |
|  |  |  |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | One of the defenses against SQL injection is using PreparedStatements. |

| **Noncompliant Code** |
| --- |
| Without PreparedStatements. |
| uName = getRequestString("username");  uPass = getRequestString("userpassword");  sql = “SELECT \* FROM Users WHERE Name = " + uName + " AND Pass = " +  uPass + ” |

| **Compliant Code** |
| --- |
| With PreparedStatements. |
| 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) Input validation for SQL injection prevention  2) Using prepared SQL statements that will prevent SQL injection  3) Using prepared SQL statements that aid in adding strength |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | MOST LIKELY | MEDIUM | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft | 2022.2 | [FIO30-C](https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+stringsFIO30-C) | The use of unfiltered data is prohibited |
| Astre’e | 22.10 |  | Data analysis |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | It’s important not to access memory that is already freed up. |

| **Noncompliant Code** |
| --- |
| Rewriting to s after it has been deleted, accessing memory that has been freed up. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| We ensure that the memory storage for s is not unassigned until AFTER it is no longer needed. |
| #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):**  1) Using static tools to identify memory issues  2) Using Quality Assurance techniques |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | MOST LIKELY | MEDIUM | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astre’e | 22.10 | Pointer use |  |
| Polyspace | 2022.2 | [MEM50-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory) | Avoid using freed memory |
| Parasoft | R2020b | [MEM50-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory) | Pointer out of bounds |
|  |  |  |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | We can use assertions to test if the pointer is NULL, if the expression is FALSE then we can assert() |

| **Noncompliant Code** |
| --- |
| Possible NULL pointer. |
| /\* 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** |
| --- |
| We can assert() to avoid printing to deallocated memory. |
| /\* assert example \*/  #include <stdio.h> /\* printf \*/  #include <assert.h> /\* assert \*/  void print\_number(int\* myInt) {  **assert (myInt!=NULL);**  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;  } |

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

| **Principles(s):**  1) Using assertions to verify code |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | MOST LIKELY | HIGH | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft | R2020b | [ERR56-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR56-CPP.+Guarantee+exception+safety) | Not fully implemented |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-008-CPP] | We must ensure that ALL exceptions are handled properly. |

| **Noncompliant Code** |
| --- |
| We have no exception handler in the function f() or main(). |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| We can add try and catch in main() to try and handle exceptions. |
| 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):**  1) Using try and blocks when implementing code  2) Continuous testing throughout the coding process |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| LOW | POSSIBILITY | MEDIUM | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astre’e | 22.10 | Catch all and catch early | Partial |
| Polyspace | 2022.2 | [ERR51-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions) | Checking for unhandled exceptions |
| Parasoft | R2020b | [ERR51-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions) | Maintaining exception handling |
|  |  |  |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | [STD-008-CPP] | We must be vigilant to close all files once we no longer need access to them. |

| **Noncompliant Code** |
| --- |
| Here we use open(), but we never call close() to the file. |
| #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** |
| --- |
| We make sure we called close() to avoid the std::terminate |
| #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):**  1)Executing the best practice of closing open files when complete  2)Closing open files provides strength |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| MEDIUM | NOT LIKELY | MEDIUM | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 5.4 | ALLOC.LEAK | Leak |
| Parasoft | R2020b | [FIO51a-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed) | Will make sure resources are freed up |
| Polyspace | 2022.2 | [FIO51a-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed) | Check for possible resource leak |
|  |  |  |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | [STD-009-CPP] | Checking validity of iterator ranges. |

| **Noncompliant Code** |
| --- |
| We use invalid values for the for\_each() func. while our 2 iterators are !=, c.end will keep moving forward. |
| #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** |
| --- |
| Correctly passed the 2 iterators. |
| #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):**  1) Protection of iterators to prevent overflows  2) Making sure to define iterator range  3) The prevention of iterator overflows adds strength |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | POSSBILE | HIGH | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astre’e | 22.10 |  |  |
| Parasoft | 2022.2 | [CTR53a/b-CPP](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR53-CPP.+Use+valid+iterator+ranges) | Prevention of using an iterator in a range that doesn’t exist |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| INT | [STD-010-CPP] | We must guarantee that when we use our signed ints our operations don’t cause overflow issues. |

| **Noncompliant Code** |
| --- |
| Overflow result from this addition operation. |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| We can test for overflow situations prior to the operations. |
| **#include <limits.h>**    void f(signed int si\_a, signed int si\_b) {  signed int sum;  **if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||**  **((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {**  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  /\* ... \*/  } |

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

| **Principles(s):**  1) Input validation can cause overflows  2) Testing calculations to prevent INT overflows |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| HIGH | MOST LIKELY | HIGH | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astre’e | 22.10 | INT overflow | Checked |
| Polyspace | 2022.2 | [INT32a/b/c-C](https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow) | Avoiding INT overflows |
| Parasoft | R2020b | [INT32a/b/c-C](https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow) | Untested Calculations |
|  |  |  |  |

### 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.

We will add security tool training in to the Assess and Plan portion of the DevOps infrastructure, Providing training on IDE security measures in the Design and Build portions of the map. Then, adding static testing assessments in the Verify and Test phase.

### 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 | LOW | NOT LIKELY | LOW | P3 | L2 |
| STD-002-CPP | HIGH | MOST LIKELY | MEDIUM | P12 | L1 |
| STD-003-CPP | HIGH | MOST LIKELY | MEDIUM | P18 | L1 |
| STD-004-CPP | HIGH | MOST LIKELY | MEDIUM | P18 | L1 |
| STD-005-CPP | HIGH | MOST LIKELY | MEDIUM | P18 | L1 |
| STD-006-CPP | HIGH | MOST LIKELY | HIGH | P9 | L2 |
| STD-007-CPP | LOW | NOT LIKELY | MEDIUM | P4 | L3 |
| STD-008-CPP | MEDIUM | NOT LIKELY | MEDIUM | P4 | L3 |
| STD-009-CPP | HIGH | LIKELY | HIGH | P6 | L2 |
| STD-010-CPP | HIGH | MOST LIKELY | HIGH | P9 | L2 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
|  |  |  |  |  |  |

### Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is designed to prevent the attacker from accessing the unencrypted data by ensuring the data is encrypted when on disk. If an attacker obtains a hard drive with encrypted data but not the encryption keys, the attacker must defeat the encryption to read the data.  Finding encryption tool and implementing them to protect the data from being breached. |
| Encryption at flight | Encryption in flight is the encryption of data that moves over a network. This is especially important for those using open internet for transporting data, which is part of most public cloud implementations.  Using secure network protocols to insure a base layer of network attacks. |
| Encryption in use | In-Use encryption takes a new approach that ensures that sensitive data is never left unsecured, regardless of or lifecycle stage (at rest, in transit, or in use) source, or location (on premise, cloud, or hybrid).  There are cryptographic tools that can be used to protect data during calculations. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication allows us to verify user credentials. By using username and passwords verification, ensuring ALL users are verified. |
| Authorization | Permissions = Authorization. Defining roles for each individual ensures only has access to what they are needed to use. Different roles require different permissions. |
| Accounting | Accounting means keeping records of any kind of transaction. Including, logins or profile creation. Accounting allows us to validate what users are doing and when. |

**\***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 | 01/21/23 | Initial Template | Chase Carney |  |
| 2.0 | 02/11/23 | Template Completion | Chase Carney |  |
|  |  |  |  |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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