**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 | Making sure inputs into the system are validated and checked to prevent malicious attacks such as injection attacks. |
| 1. Heed Compiler Warnings | Paying attention to warnings the IDE gives to prevent things like memory leaks and other problems that could cascade into bigger issues if not solved. |
| 1. Architect and Design for Security Policies | Designing from the start to implement security that is needed and braking up the program in order to keep security as high as possible. |
| 1. Keep It Simple | If code can be made to run more efficiently while having less code its generally better. For example if a loop can be run through less times its better than running through long lists of data repeatedly. |
| 1. Default Deny | Refuses access to data unless they have the appropriate credentials. |
| 1. Adhere to the Principle of Least Privilege | If your system doesn’t need access to it, you shouldn’t request access. Having a simple program asking for more than it needs can lead to security issues with hackers. |
| 1. Sanitize Data Sent to Other Systems | When transferring data to say a SQL sever you want to make sure its clear of malicious coded that could be used to run commands from unwanted users. |
| 1. Practice Defense in Depth | More defense is better if one layer of protections fails there should be other layers to protect it. |
| 1. Use Effective Quality Assurance Techniques | Testing your code before it goes public is important to make sure you can prevent attacks you might have seen. Using 3rd parties and having peer reviews are great ways to test and check code before publishing it. |
| 1. Adopt a Secure Coding Standard | Having a standard is important when working with a team to make sure everyone is using similar coding practices and making sure everything is properly secure on all fronts. |

### 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 | Ensure float conversions via cast aren’t accidental. |

| **Noncompliant Code** |
| --- |
| When casting we want to make sure that data isn’t lost. This can happen due to rounding when casting a float to an int. |
| void func(void) {  float num1 = 123456789.123456;  int num2 = num1;  } |

| **Compliant Code** |
| --- |
| To make the code compliant we static cast the float to ensure the correct information is transferred. |
| void func(void) {  float num1 = 123456789.123456;  int num2 = static\_cast<int>(num1);  } |

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

| **Principles(s):** 1, This standard applies to the principal, validate input data, as it ensures conversion has the proper input before casting. Failure to do so could cause buffer overflow and crash the program. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Priority | High | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppchek | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Parasoft C/C++test | 2023.1 | CERT\_C-INT31-i | Avoid Integer overflows |
| Converty | 2017.07 | NEGATIVE\_RETURNS | Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted |

#### 

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Ensure operations don’t result in overflow. |

| **Noncompliant Code** |
| --- |
| Using this method doesn’t account for the sum of a and b overflowing during the operation. |
| void func(signed int a, signed int b) {  signed int sum = a + b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Code that complies checks to make sure the operation wont overflow and handles the error. |
| #include <limits.h>    void f(signed int a, signed int b) {  signed int sum;  if (((b > 0) && (a > (INT\_MAX - b))) ||  ((b < 0) && (a < (INT\_MIN - b)))) {  /\* Handle error \*/  } else {  sum = a + b;  }  /\* ... \*/  } |

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

| **Principles(s):** 1,2 Practice defense in depth and validate input data, assumptions shouldn’t be made about data |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 | integer-overflow | Fully Checked |
| CodeSonar | 8.1p0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW  MISC.MEM.SIZE.ADDOFLOW  MISC.MEM.SIZE.BAD  MISC.MEM.SIZE.MULOFLOW  MISC.MEM.SIZE.SUBUFLOW | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Helix QAC | 2024.1 | C2800, C2860  C++2800, C++2860  DF2801, DF2802, DF2803, DF2861, DF2862, DF2863 |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Ensure strings have adequate space for characters and the null terminator. |

| **Noncompliant Code** |
| --- |
| The input is unbound and could lead to buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Using std::string ensures the data isn’t truncated and is great against buffer overflow. |
| #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, 2, 3, 4 validation, simplicity, and security are very important for this. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.04 |  | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR31 | Detects calls to unsafe string function that may cause buffer overflow  Detects potential buffer overruns, including those caused by unsafe usage of fscanf() |
| CodeSonar | 8.1p0 | LANG.MEM.BO  LANG.MEM.TO  MISC.MEM.NTERM  BADFUNC.BO.\* | Buffer overrun  Type overrun  No space for null terminator  A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Utilize parameters over raw string queries to access data sources. |

| **Noncompliant Code** |
| --- |
| This code is non-compliant as it injects raw strings into the query engine and can allow users to enter additional items in the string and cause it to fail. |
| #include <iostream>  #include <sqlite3.h>  void func()  {  sqlite3\* DB;     int exit = 0;     exit = sqlite3\_open("example.db", &DB);  string sql("INSERT INTO PERSON VALUES(1, 'STEVE', 'GATES', 30, 'PALO ALTO', 1000.0);"       exit = sqlite3\_exec(DB, sql.c\_str(), NULL, 0, &messaggeError);     if (exit != SQLITE\_OK) {         std::cerr << "Error Insert" << std::endl;         sqlite3\_free(messaggeError);     }  } |

| **Compliant Code** |
| --- |
| This code is compliant as it uses parameters instead of raw strings to insert data into the query. Its much harder to succumb to SQL injection style attacks. |
| #include <iostream>  #include <sqlite3.h>  void func()  {  sqlite3\* DB;     int exit = 0;     exit = sqlite3\_open("example.db", &DB); |

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

| **Principles(s):** 3, 7 we are trying to prevent a specific kind of attack and we don’t send data to another system that could compromise data. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Priority | Medium | 5 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| Findbugs | 2.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |

#### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| S is deleted then referenced again which can be used as a point of attack to run code with the variables permission. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| Making sure you only delete allocated memory only after you are done with it is important to ensure security. |
| #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):** 4, 8 simplicity and defense in depth are what apply to this section |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM50 |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 8.1p0 | ALLOC.UAF | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Don’t leave in assertion code |

| **Noncompliant Code** |
| --- |
| Even when turned off assertion code can still be used for malicious attacks. |
| void f()  {  std::vector<int> myVec\* = std::vector<int>(5);  assert(myVec != nullptr);  myVec.push\_back(1);  } |

| **Compliant Code** |
| --- |
| Removing assertion code completely is the best way to ensure security. |
| void f()  {  std::vector<int> myVec\* = std::vector<int>(5);  myVec.push\_back(1);  } |

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

| **Principles(s):** 3, 4, 6, 8 apply to design, simplicity, least privilege, and defense in depth. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | 1 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |

|  |  |  |  |
| --- | --- | --- | --- |
| SonarQube | 6.7 | S3346 | Expressions used in "assert" should not produce side effects |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| Because there is no handling of the exception the program will be terminated |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| The main function has a handle point which will allow the exception to be handled and the program wont terminate. |
| 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, 3, 4, 8. These principles apply because of the relation to input validation, designing for security, simplicity, and defense in depth. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | main-function-catch-all  early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 |  |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Unreachable Catch | 2024.1 | C++4035, C++4036, C++4037 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exceptions | STD-008-CPP | Do not leak resources when handling exceptions. |

| **Noncompliant Code** |
| --- |
| The code doesn’t handle cleaning up myVector |
| #include <vector>  void f() {  std::vector<int> \*myVector = new std::vector<int>(5);  try  {  myVector->at(1)  } catch (…) {  // Process error with logic  throw;  }  delete myVector;  } |

| **Compliant Code** |
| --- |
| The new code destroys the allocated vector even if an error is thrown. |
| #include <vector>  void f() {  std::vector<int> \*myVector = new std::vector<int>(5);  try  {  myVector->at(1)  } catch (…) {  // Process error with logic  throw;  }  finally  { |

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

| **Principles(s):** 2, 9 listening to warnings and using quality checks will make sure the program doesn’t crash or have issues. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | High | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | |  |  | | --- | --- | |  | 8.1p0 | | **ALLOC.LEAK** | Leak |
| Helix QAC | 2024.1 | DF4756, DF4757, DF4758 |  |
| Klocwork | 2024.1 | CL.MLK  MLK.MIGHT  MLK.MUST  MLK.RET.MIGHT  MLK.RET.MUST  RH.LEAK |  |
| LDRA tool suite | |  |  | | --- | --- | |  | 9.7.1 | | 50 D | Partially implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | STD-009-CPP | Use valid iterator ranges |

| **Noncompliant Code** |
| --- |
| This loop can result in a range going beyond what has been defined because it iterators before checking. |
| #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** |
| --- |
| The code passes the iterations in the correct order making sure it wont run into issues out of range. |
| #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):** 10 this applies to adopt as clear coding standard as it effects all code that should be followed rather than a specific point. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | overflow\_upon\_dereference |  |
| CodeSonar | 8.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TO  LANG.MEM.TU  LANG.MEM.TBA  LANG.STRUCT.PBB  LANG.STRUCT.PPE  LANG.STRUCT.PARITH | Buffer overrun Buffer underrun Type overrun Type underrun Tainted buffer access Pointer before beginning of object Pointer past end of object Pointer Arithmetic |
| Helix QAC | 2024.1 | C++3139, C++3140  DF2891 |  |
| Klocwork | 2024.1 | ABV.ANY\_SIZE\_ARRAY  ABV.GENERAL  ABV.GENERAL.MULTIDIMENSION  ABV.STACK  ABV.TAINTED  SV.TAINTED.ALLOC\_SIZE  SV.TAINTED.CALL.INDEX\_ACCESS  SV.TAINTED.CALL.LOOP\_BOUND  SV.TAINTED.INDEX\_ACCESS |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | STD-010-CPP | Do not access closed files |

| **Noncompliant Code** |
| --- |
| Stdout stream is trying to access a file already closed |
| #include <stdio.h>    int close\_stdout(void) {  if (fclose(stdout) == EOF) {  return -1;  }    printf("stdout successfully closed.\n");  return 0;  } |

| **Compliant Code** |
| --- |
| Once the file is closed it no longer tries to access it. |
| #include <stdio.h>    int close\_stdout(void) {  if (fclose(stdout) == EOF) {  return -1;  }    fputs("stdout successfully closed.", stderr);  return 0;  } |

**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 | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 23.0 |  | Supported |
| CodeSonar | 8.1p0 | IO.UAC | Use after close |
| Compass/ROSE |  |  |  |
| Converity | 2017.07 | USE\_AFTER\_FREE | Implemented |

### 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.

Starting on program security as early as possible is of the most importance to DevSecOps. Starting the security process in the access and planning phase allows programmers to have a similar understanding of security threats a program will face and how to program to combat these problems. Having best practice for designing and building helps make testing and verifying code much quicker with security threats already being taken care of in the building process. Following this is launching the program where more security tests can be run, and security experts can try and penetrate the code and gain access to vulnerabilities. While all this does make an app more secure monitoring and detection is still essential and new vulnerabilities and be found. In the event that someone does gain access to a system they should have a ready response team is also important to lock down and patch up any vulnerabilities that might have been missed. This leads to maintaining and stabilizing a system after an attack to make sure more information isn’t taken and now the app can be brought back to access and plan to regularly keep up with potential weak points and stay ahead of unwanted users.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Likely | High | High | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Not Likely | Low | Low | 1 |
| STD-007-CPP | Low | Probable | Medium | Low | 3 |
| STD-008-CPP | Low | Priority | High | Low | 3 |
| STD-009-CPP | High | Probable | High | Medium | 2 |
| STD-010-CPP | Medium | Unlikely | Medium | Low | 3 |

### 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 at rest | When storing passwords they should be encrypted to prevent unwanted users who gain access the ability to read or use them. Encryption should be updated as the technology is updated along with using the correct encryption for the situation. |
| Encryption in flight | When transferring data over a connection a proper SSL/TLS connection should be made between the webserver and database to ensure the information isn’t intercepted. Additional measures such as a VPN can be used to improve security even further. |
| Encryption in use | Using techniques like protected memory is important to make sure any information that is being used for computation or delivering data to the customer. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | When accessing information, a user should be required to validate they are who they are by means of passwords or more current technology with facial identification or thumbprint. The more unique a user’s validation method is the harder it is to replicate and steal. Further device or IP remembrance can further assist this by ensuring abnormal logins are further verified. |
| Authorization | After the system determines you are who you are next it assigns you the information that you have access to too. For a banking app this could be your banking information and only your banking information. If you work for the bank, you might be able to access more accounts than you are authorized to use. Using practices like default deny is a good practice to make sure someone who shouldn’t have access to something doesn’t. |
| Accounting | Logging information that the users access, or changes is also a good way to ensure someone who shouldn’t have access to something doesn’t. If unwanted users gain access you’ll be able to see everything they inputted or saw to get into the system along with what the user was after in order to make It more secure. |

**\***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 |  |
| 1.1 | 03/31/2024 | Milestone 3 completion | Dominic Aguirre | [Insert text.] |
| 1.2 | 04/14/2024 | Project 1 completion | Dominic Aguirre | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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