**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 inputting data from the user ensure that the data can is properly formatted when it enters the system. This will act as a method of injection prevention. It is important to do input validation on the client and the server, especially with login information. The client side will do a check to make sure the data is properly formatted, and the server side will verify that the information be sent or requested belongs in the system. |
| 1. Heed Compiler Warnings | Even though the program compiles with warnings it is important to investigate each warning to ensure that it does not open a new vulnerability. Just because the program runs does not mean it is safe. |
| 1. Architect and Design for Security Policies | Design software to implement and enforce security policies. If the bones of the software force users and other developers to follow the security policies, it is harder for them to be ignored. |
| 1. Keep It Simple | Keep the design as simple and small as possible. The simpler it is, the easier it is to understand and the smaller it is, the less vulnerabilities that could be present. |
| 1. Default Deny | The default privilege is no privilege. Access is only given to users who absolutely need it, everyone else gets no access. |
| 1. Adhere to the Principle of Least Privilege | The people who do gain privilege get the absolute minimum they need to complete their tasks. Each user has access to the absolute smallest percentage of the system as possible. |
| 1. Sanitize Data Sent to Other Systems | When passing data between subsystems make sure that the data that is sent is only what is needed. This prevents injection attacks between different internal subsystems. |
| 1. Practice Defense in Depth | Make multiple redundant security layers throughout the system to ensure that if one layer fails there are more layers protecting the system. |
| 1. Use Effective Quality Assurance Techniques | Use these techniques to identify and eliminate vulnerabilities, examples are fuzz testing, penetration testing, and source code audits. These methods should be used in combinations to discover and eliminate vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Develop a coding standard for everyone to follow that is working on the same language and platform. This ensures that anyone at the organization can read another person’s code quickly and easily. |

### 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** | STR51-CPP | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer and failure, this code can lead to undefined behavior when the environment variable does not exist. |
| #include <cstdlib>  #include <string>  Void f() {  Std::string tmp(std::getenv(“TMP”));  If (!tmp.empty()) {  // ….  }  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the results from the call to std::getenv() are checked for null before the std::string object is constructed |
| #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.**

| **Encryption in Use: This does not directly map to any of the standards, but it most closely maps to encryption in use. When using data, it is important to create strings from the correct location and ensure that all pointers that are being used have been initialized to the correct location.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Assert\_failure |  |
| CodeSonar | 7.4p0 | LANG.MEM.NPD | Null pointer Dereference |
| Helix QAC | 2023.3 | DF470, DF4771, DF4772, DF4773, DF4774 |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT31-C | Ensure that integer conversions do not result in lost or misinterpreted data. |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss oof data(truncation)\_ and loss of sign(sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type. This noncompliant code example results in a truncation error on most implementations |
| #include <limits.h>  Void func(void) {  Unsigned long int u\_a = ULONG\_MAX;  Signed char sc;  Sc = (signed char)u\_a; // casting eliminates warning  } |

| **Compliant Code** |
| --- |
| Validate ranges when converting from an unsigned type to a signed type. This compliant solution can be used to insert a value of unsigned long int type of a value of signed char type. |
| #include <limits.h>  Void func(void) {  Unsigned long int u\_a = ULONG\_MAD;  Signed char sc;  If (u\_a <= SCHAR\_MAX) {  Sc = (signed char)u\_a;  } else {  // Error  }  } |

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

| **Encryption in use: This does not directly map to the standard. However, it is important that when we are using data integers are converted properly to ensure that pointers remain pointing to the correct location and that values are not corrupted.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 |  | Supported via MISRA C:2013 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| CodeSonar | 7.4p0 | LANG.CAST.PC.AV  LANG.CAST.PC.CONST2PTR  LANG.CAST.PC.INT  LANG.CAST.COERCE  LANG.CAST.VALUE  ALLOC.SIZE.TRUNC  MIST.MEM.SIZE.TRUNC  LANG.MEM.TBA | Cast: arithmetic type/void pointer  Conversion: integer constant to pointer  Conversion: pointer/integer  Coercion alters values  Cast alters values  Truncation of allocation size  Truncation of size  Tainted buffer access |
| Coverity | 2017.07 |  | Can detect vilations of this rule. However, false warning may be raised if limits.h is included |
| Cppcheck | 2023.3 | C2850, C2855, C2890, C2895, C2900, C2905  C++2850, C++2855, C++2890, C++2895, C++2900, C++2905, C++3000, C++3010  DF2851, DF2852, DF2853, DF2856, DF2857, DF2858, DF2891, DF2892, DF2893, DF2896, DF2898, DF2901, DF2902, DF2903, DF2906, DF2907, DF2908 | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR32-C | Do not pass a non-null-terminated character sequence to a library function that expects a string. |

| **Noncompliant Code** |
| --- |
| This code example is noncompliant because the character sequence c\_str will not be null-terminated when pass as an argument to printf(). |
| #include <stdio.h>  Void func(void) {  Char c\_str[3] = “abc”;  Printf(“%s\n”, c\_str);  } |

| **Compliant Code** |
| --- |
| This compliant solution does not specify the bound of the character array in the array declaration. If the array bound is omitted, the compiler allocates sufficient storage to store the entire string literal, including the terminating null character. |
| #include <stdio.h>  Void func(void) {  Char c\_str[] = “abc”;  Printf(“%s\n”, c\_str);  } |

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

| **N/A** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 |  | Supported  Astree supports the implementation of library stubs to fully verify this guideline |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR32 | Partially implemented: can detect some violation of the rule |
| CodeSonar | 7.4p0 | MISC.MEM.NTERM.CSTRING | Unterminated C String |
| Coverity | 2017.07 | STRING\_NULL | Fully implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | IDS00-J | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| The JDBC library provides an API for building SQL commands that sanitize untrusted data. The java.sql.PreparedStatement class properly escapes input strings, prevent SQL injection when used correctly. This code example modifies the doPrivilegedAction() method to use a Prepared Statement instead of java.sql.Statement. However, the prepared statement still permits a SQL injection attack by incorporating the unsensitized input argument username into the prepared statement. |
|  |

| **Compliant Code** |
| --- |
| This compliant solution uses a parametric query with a ? character as a placeholder for the argument. This code also validates the length of the username argument, preventing an attacking from submitting an arbitrarily long user name. |
|  |

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

| **Data in flight: When receiving data for input it is important that the data received is encrypted. It is also important to sanitize the data once it is received. Even if it was processed on the user side, they might be a bad actor and it needs to be verified on the server side as well.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors(see Chapter 8) |
| CodeSonar | 7.4p0 | JAVA.IO.INJ.SQL | SQL Injection(java) |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | EXP33-C | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the set\_flag() function is intended to set the parameter, sign\_flag, to the sign of number. However, the programmer neglected to account for the case where number is equal to 0. Because the local variable sign is uninitialized when calling the set\_flag() and is never written to by set\_flag(), the comparison operation exhibits undefined behavior when reading sign. |
|  |

| **Compliant Code** |
| --- |
| This compliant solution trivially repairs the problem by accounting for the possibility that number can be equal to 0.  Although compilers and static analysis tools often detect uses of uninitialized variables when they have access to the source code, diagnosing the problem is difficult or impossible when either the initialization or the use takes place in object code for which the source code is inaccessible. Unless doing so is prohibitive for performance reasons, an additional defense-in-depth practice worth considering is to initialize local variables immediately after declaration. |
|  |

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

| **Data in use: It is important that the data we are using is in an initialized block of memory. If we try to reference data that was placed in an uninitialized block of memory undefined behavior will occur.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | Uninitialized-local-read  Uninitialized-variable-use | Fully checked |
| Axion Bauhaus Suite | 7.2.0 | CertC-EXP33 |  |
| CodeSonar | 7.4p0 | LANG.MEM.UVAR | Uninitialized Variable |
| Compass/Rose |  |  | Automatially detects simple violations of this rule, although it may return some false positives. It may not catch more complex violations, such as initialization within functions taking unitialized variables as arguments. It does catch the second noncompliant code example, and can be extended to catch the first as well. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | MSC60-J | Do not use assertions to verify the absence of runtime errors. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses the assert statement to verify that input was available. |
| BufferedReader br;  String line;  Line = br.readLine();  Assert line != null; |

| **Compliant Code** |
| --- |
| This compliant solution demonstrates the recommended way to detect and handle unavailability of input. |
| BufferedReader br;  String line;  Line = br.readLine();  If (line == null) {  // Handle error  } |

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

| **N/A** |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| N/A |  |  |  |  |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Jtest | 2023.1 | CERT.MSC60.ASSERT | Do not use assertions in production code |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-J | Prefer user-defined exceptions over more general exception types. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example attempts to distinguish between different exceptional behavior Aby looking at the exceptions message. |
|  |

| **Compliant Code** |
| --- |
| This compliant solution uses specific exception types and defines new special-purpose exception types where required. |
|  |

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

| **N/A** |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| N/A |  |  |  |  |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft Jtest | 2023.1 | CERT.ERR51.NCE | Do not catch exception types which are too general or unchecked exception |
| SonarQube | 9.9 | S1193 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Sensitive Information | MSC03-J | Never hard code sensitive information |

| **Noncompliant Code** |
| --- |
| This noncompliant code example includes a hard-coded server IP address in a constant string. A malicious user can use the javap -c Ipaddress command to disassemble the class and discover the hard-coded server IP address. |
|  |

| **Compliant Code** |
| --- |
| This compliant solution retrieves the server IP address from an external file located in a secure directory, as recommended by FIO00-J. It reads the file in compliance with FIO10-J. Exposure of the IP address is further limited by storing it in a char array rather than a java.lang.String, and by clearing the server IP address from memory immediately after user. |
|  |

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

| **Data at rest: Data at rest should never be placed in the source code. All data should be placed in the DB system and stored in physically protected locations. It should also be stored on disks that are encrypted and preferably the data will never need to be decrypted.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Codesonar | 7.4p0 | JAVA.HARDCODED.PASSWD  JAVA.MISC.SD.EXT | Hardcoded Password(java)  Sensitive Data Written to External Storage(java) |
| Coverity | 7.5 | HARDCODED\_CREDENTIALS  CONFIG  FB.DMI\_CONSTANT\_DB\_PASSWORD  FB.DMI\_EMPTY\_DB\_PASSWORD | Implemented |
| Fortify | 1.0 | Password\_Managment  Password\_Managment\_Hardcoded\_Password | Partially implemented |
| PMD | 1.0 | AvoidUsingHardCodedIP | Partially Implemented |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Accessing files | FIO51-CPP | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a std::fstream object file is constructed. The constructor for std::fstream calls the std::basic\_filebuf<T>::open(), and the default std::teminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed. |
|  |

| **Compliant Code** |
| --- |
| In this compliant solution, std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
|  |

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

| **Data in use: Once we are done using data it is important that the file is closed. This is because data in use is often in memory and it is easier to access when being used. Once done with the data it can be returned to the DB system and remain on an encrypted disk where access is more difficult.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2023.3 | DF4786, DF4787, DF4788 |  |
| Klocwork | 2023.3 | RH.LEAK |  |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Handling | ERR50-CPP | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func() may throw an exception. |
|  |

| **Compliant Code** |
| --- |
| In this compliant solution, f() handles all exceptions thrown by throwing\_func() and does no rethrow. |
|  |

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

| **Data in use: When a program is terminated abruptly data in use can be lost or corrupted. This is why it is important to not terminate abruptly, even in the event of a crash. When the system is terminating or there is a crash, it needs to be detected and follow a proper shut down procedure that involves returning all data in use or in flight.** |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Stdlib-use | Partially checked |
| CodeSonar | 7.4p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| Helix QAC | 2023.3 | C++5014 |  |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced enforcement |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation should be in place for pre-production and production environments. When writing pre-production code, automation should be placed in the code throughout the development process to ensure that the testing phase runs smoothly. Placing the automation in early will also help catch simple mistakes made throughout development. It is worth noting that the automated checks for asserts should not be placed in Pre-production code.

In production environments, most of the automated checks should already be implemented from the pre-production environment. However, some new checks will need to be added, this should happen immediately to ensure that the code base is ready for a production release.

### 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 |
| --- | --- | --- | --- | --- | --- |
| STR51-CPP | High | Unlikely | Medium | P6 | L2 |
| STR51-CPP | High | Likely | Medium | P18 | L1 |
| INT31-C | High | Probable | High | P6 | L2 |
| STR32-C | High | Probable | Medium | P12 | L1 |
| IDS00-J | High | Likely | Medium | P18 | L1 |
| EXP33-C | High | Probable | Medium | P12 | L1 |
| N/A |  |  |  |  |  |
| N/A |  |  |  |  |  |
| MSC03-J | High | Probably | Medium | P12 | L1 |
| FIO51-CPP | Medium | Unlikely | Medium | P4 | L3 |
| ERR50-CPP | Low | Probably | Medium | P4 | L3 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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 for data that is stored in a physical or digital medium, such as a hard drive or file cabinet. The goal is to prevent an attacker from being able to access the data in the event it is stolen. Some examples are full disk encryption, file-level encryption, and database encryption. |
| Encryption at flight | Encryption at flight is for data that is currently being transferred (in flight). This can be in network, or out of network. In network is not as large of a security risk as out of network. Some security layers that should be implemented to protect this data are SSL and TLS. |
| Encryption in use | Encryption in use is for protecting data that is currently being used by the application, this is often data that is stored in memory. The goal is for data to remain confidential while processing, this makes the application more secure when third parties are handling the data. One example of how to accomplish this is to use homomorphic encryption, a type of encryption that allows for computations to be performed without the need for decryption first. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is verifying that the user attempting to access the service is who they say they are. To implement this several methods should be involved, examples include biometrics, passwords, and smart cards (for physical access). Multi-factor authentication should always be used for secure authentication. Note: SMS as a second factor of authentication is not very secure, email is always preferred. |
| Authorization | Authorization can also be referred to as access control, it is the practice of limiting what each user is able to access. It is best to follow a policy of no access, meaning that each user by default has access to nothing. A common form of access control is role-based access where an administrator can define a set of roles and permissions for each role. Individual users are then assigned a role that defines the access they are allowed to the system. |
| Accounting | Accounting is the process of tracking and recording activities that occur on the system. This can be logins, data transfers, requests, etc. To implement an accounting system, logs will need to be created that will automatically store information. It may be a good idea to have separate logs for different locations, types of actions, etc. |

**\***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 | 11/2023 | Partially Implemented | Hatcher Blair |  |
| 2.0 | 12/2/2023 | Fully Implemented | Hatcher Blair |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

| Language | Acronym |
| --- | --- |
| C++ | CPP |
| C | C |
| Java | J |