**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 | Validating input is important when considering vulnerabilities and a rule to go by is not to trust a user’s input. Validating input data makes sure there is not buffer overflow, memory altering, wrap arounds, etc. |
| 1. Heed Compiler Warnings | Waring from a compiler warning a developer from possible vulnerabilities like wrap arounds and buffer overflows to name a few. |
| 1. Architect and Design for Security Policies | The understanding memory architecture design of how values are used and secure ways to properly use the design from vulnerabilities. Understanding the architect helps the developers understand how values will execute and behave. |
| 1. Keep It Simple | Keeping it simple helps a developer and others to understand blocks of code and perform better. Complex code can increase the likelihood of errors |
| 1. Default Deny | Access grants base on permission which means denial may occur depending of the standards provide by the owner(s). |
| 1. Adhere to the Principle of Least Privilege | Process executed with the least set of privilege necessary to do a job which should be accessed for the least amount of time required to complete privileged task |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data that is passed to complex subsystems and invoking the process of data sanitizing for strengthening strategies to attacks. |
| 1. Practice Defense in Depth | The understanding of how programs vulnerabilities work and understanding how to fix them with secure development and practicing it in the real world. Managing risk with multiple defense strategies. |
| 1. Use Effective Quality Assurance Techniques | The use of effective quality assurance techniques ensures the quality process in the life cycle for development. |
| 1. Adopt a Secure Coding Standard | Standards for coding vulnerabilities for securing them. Standards for memory, user input, data management, program runtime all to secure them from risk. Applying the secure coding standards |

## 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** | INT30-C | Ensure that unsigned integer operations do not wrap |

| **Noncompliant Code** |
| --- |
| Here a wrap around can be caused with he addition of ui\_a and ui\_b that can also allocate insufficient memory |
| void func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum = ui\_a + ui\_b;    /\* ... \*/  } |
|  |

| **Compliant Code** |
| --- |
| In the compliant code there is a precondition test with the if statement that sets a UINT\_MAX |
| #include <limits.h>    void func(unsigned **int** ui\_a, unsigned **int** ui\_b) {    unsigned **int** usum;    if (UINT\_MAX - ui\_a < ui\_b) {      /\* Handle error \*/    } else {      usum = ui\_a + ui\_b;    }    /\* ... \*/  } |
|  |

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

|  |
| --- |
| **Principles(s):**Best Practices general for the best way of accepting to perform a task. To this we apply the best practices by learning the way these task work in the development process. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | Integer\_overflow | Implemented |
| Klocwork | 2018 | Num.overflow  Cwarn.noeffect.outofrange |  |
| LDRA tool  Suite | 9.7.1 | 493 S, 494 S | Partially implemented |
| Parasoft | 2020.2 | Cert c-int30-a  Cert c-int30-b | Avoid integer overflow |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | EXP53-CPP | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| In this noncompliant code we experience an undefined behavior from the uninitialized local variable to print its value |
| #include <iostream>    void f() {  **int** i;    std::cout << i; |

| **Compliant Code** |
| --- |
| The compliant code here we initialize prior to the print the value by adding the = o; |
| #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):** polices and standards. For what we are trying to achieve and how we are trying to achieve by initializing the variable. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Uninitialized read | Partially checked |
| Clang | 3.9 | Wuninitialezed  Clan-analyzer-cor.UndefinedBinaryOperatorResult | Does not catch all instances of this rule such as uninitialized values read from heap allocated memory |
| Klockwork | 2018 | Uninit.ctor.might  Uninit.ctor.must |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STT50-CPP | Guarantee that storage for string has sufficient space for character data and the null terminator |

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

| **Compliant Code** |
| --- |
| One of the best way to prevent buffer overflow as in this example is to use std::string instead of a bounded array |
| #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):** polices and standards. For what we are trying to achieve and how we are trying to achieve by initializing the variable. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clockwork | 2018 | Nnts.might  Nnts.tainted |  |
| LDRA tool suite | 9.7.1 | 489 s 66x 70x 71s | Partially implemented |
| Sonarqube | 4.10 | S3519 |  |
| Codesonar | 5.4p0 | Misc.mem.nterm  Lang.mem.bo  Lang.mem.to | No space for null terminator  Buffer overrun  Type overrun |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | IDSOO-J | Preventing sql injection |

| **Noncompliant Code** |
| --- |
| In the noncompliant code there is an unsanitized input argument username into the sql command that allow the attacker use the sql injection |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;    class Login {    public Connection getConnection() throws SQLException {      DriverManager.registerDriver(new              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"      return DriverManager.getConnection(dbConnection);    }      String hashPassword(char[] password) {      // Create hash of password    }      public void doPrivilegedAction(      String username, char[] password    ) throws SQLException {      Connection connection = getConnection();      if (connection == null) {        // Handle error      }      try {        String pwd = hashPassword(password);        String sqlString = "select \* from db\_user where username=" +          username + " and password =" + pwd;        PreparedStatement stmt = connection.prepareStatement(sqlString);          ResultSet rs = stmt.executeQuery();        if (!rs.next()) {          throw new SecurityException("User name or password incorrect");        }          // Authenticated; proceed      } finally {        try {          connection.close();        } catch (SQLException x) {          // Forward to handler        }      }    }  } |

| **Compliant Code** |
| --- |
| Here they use a length to prevent a user from inputting long code and ? which is used as a compliant solution as a placeholder for the argument. |
| public void doPrivilegedAction(    String username, char[] password  ) throws SQLException {    Connection connection = getConnection();    if (connection == null) {      // Handle error    }    try {      String pwd = hashPassword(password);        // Validate username length      if (username.length() > 8) {        // Handle error      }        String sqlString =        "select \* from db\_user where username=? and password=?";      PreparedStatement stmt = connection.prepareStatement(sqlString);      stmt.setString(1, username);      stmt.setString(2, pwd);      ResultSet rs = stmt.executeQuery();      if (!rs.next()) {        throw new SecurityException("User name or password incorrect");      }        // Authenticated; proceed    } finally {      try {        connection.close();      } catch (SQLException x) {        // Forward to handler      }    }  } |

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

|  |
| --- |
| **Principles(s):** best practices from he principles can be applied here because sql injections are a common this in the development world so applying this principle means using best practices to prevent these. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The checker frame | 2.1.3 | Tainting checker | Trust and security error |
| Parasoft | 2020.2 | Bd security tdsql | Protect against sql injecitons |
| Sonarqube | 6.7 | 52077  S3649 | Execution sql queries is security sensitive |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM50-CPP | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| In the noncompliant code s is dereferenced after is has been deallocated. Is this results in a write-after-free there is a vulnerability may be exploit to run arbitrary code with the permission of vulnerable process. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| In the compliant code the memory is not deallocated until it is no longer required with new and delete. |
| #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):** position papers which involves documentations can be a good principle here since we can documentation usage for better secure over memory usage that can be documented |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Splint | 5.0 | [Insert text.] |  |
| PVS-studio | 7.07 | V586, v774 |  |
| LDRA tool suite | 9.7.1 | 483 s 484 s | Partially implemented |
| PRQA QA C++ | R202a | Cert c++ meme50-cpp | Check for  Pointer access out of bounds  Deallocation of previously deallocated pointers  Use of previously freed pointers |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | DCLO3-C | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| Here the noncompliant code the assert() macro to assert concerning memory |
| #include <assert.h>    struct timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    **int** func(void) {  **assert**(sizeof(struct timer) == sizeof(unsigned **char**) + sizeof(unsigned **int**) + sizeof(unsigned **int**));  } |

| **Compliant Code** |
| --- |
| Here the assertion is used in a padding situation for the size of unsigned int and char |
| struct timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

|  |
| --- |
| **Principles(s):** best practice and guidance principle can be applied to this file through the development for assertions for certain best practices and guidance to doc the process of the development. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion | 6.9.0 | certC-dcl03 |  |
| Clang | 3.9 | Misc-static-assert | Checked by clang tidy |
| Codesonar | 5.4p0 | Customization | User can implement a custom check that reports uses of the assert() |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully implemented |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| Here f() nor main() catch exceptions that are thrown by throwing\_func(). Std::terminate() is called here because no catch handler can be found |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| In main entry point handles all exceptions with the catch() functions for handle error |
| 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):** best practices. Here for the principles I chose best practices because using exceptions in code should be set to best practices in developing of code files. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| Axivion Bauhaus  Suite | 6.9.0 | CerC++-ERR51 | [Insert text.] |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Floating Point | FLP30-V | Do not use floating point variable as loop counter |

| **Noncompliant Code** |
| --- |
| In the noncompliant code the floating decimal number 0.1 that can not be exactly represented as a binary floating pointer number. The loop may iterate 9 or 10 times. |
| void func(void) {    for (**float** x = 0.1f; x <= 1.0f; x += 0.1f) {      /\* Loop may iterate 9 or 10 times \*/    }  } |

| **Compliant Code** |
| --- |
| In the compliant code we use an integer of 1 from which the floating point value is derived by 10.0f |
| #include <stddef.h>    void func(void) {    for (**size\_t** count = 1; count <= 10; ++count) {  **float** x = count / 10.0f;      /\* Loop iterates exactly 10 times \*/    }  } |

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

|  |
| --- |
| **Principles(s):** from the principle I think best practices and standard can adhere to this floating example by understanding the code and using best practices and standards to develop in a correct manner. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | For loop float | Fully checked |
| Axivion | 6.9.0 | Certc-flp30 | Fully implemented |
| Clang | 3.9 | Cert-flip30-c | Checked by clang-tidy |
| codeRose | 5.4p0 | Lang.struct.loop.fpc | Flaot typed loop counter |

### Coding Standard 9

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

| **Noncompliant Code** |
| --- |
| In this noncompliant code there is no call to destructors and is not properly closed. |
| #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** |
| --- |
| Here std::fstream::close() is called before the terminate is call properly closing the file resource. |
| #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):** guidance and best practices principles can be apply to this example file with the guidance of learning and knowing that closing files and how to accomplish it. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| codeSonar | 5.4p0 | ALLOC.LEAK | Leak |
| Clockwork | 2018 | RH.LEAK |  |
| Parasoft c/c++ test | 2020.2 | CERT\_CPP-FIO51-a | Ensure resources are freed |
| Parasoft insure++ |  |  | Runtime detection |

**Coding standard 10**

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declarations and  Initialized | [DCL30-C] | Declare objects with appropriate storage durations. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code the address of c\_str is assigned to variable p but it is invalid for the c\_str to go out of the scope while p holds the address that happens at the end of don’t\_do\_this. Going out of the scope of an address leaves vulnerabilities to attacks. |
| #include <stdio.h>    const **char** \*p;  void dont\_do\_this(void) {    const **char** c\_str[] = "This will change";    p = c\_str; /\* Dangerous \*/  }    void innocuous(void) {  **printf**("%s\n", p);  }    **int** main(void) {    dont\_do\_this();    innocuous();    return 0; |

| **Compliant Code** |
| --- |
| In the compliant code p is inaccessible outside the scope of string c\_str. P is declared with the same storage duration as c\_str preventing p from taking values outside this\_is\_ok |
| void this\_is\_OK(void) {    const **char** c\_str[] = "Everything OK";    const **char** \*p = c\_str;    /\* ... \*/  } |

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

|  |
| --- |
| **Principles(s):**Best Practices general for the best way of accepting to perform a task. To this we apply the best practices by learning the way these task work in the development process. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Pointered-deallocation  Return-reference-local | Fully checked |
| Axivion  Bauhaus suite | 6.90 | CertC-DCL30 | Fully implemented |
| CodeSonar | 5.4p0 | Lang.struct.rpl | Retruns pointer to local |

### Coding Standard 10

## Defense-in-Depth Illustration

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



# Project One

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

## Revise the C/C++ Standards

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

## Risk Assessment

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

## Automated Detection

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

## Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

## Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| INT30-C | High | Likely | High | Medium | L2 |
| 1 | High | Probable | Medium | Low | L1 |
| STT50-CPP | High | Likely | Medium | High | L1 |
| IDSOO-J | High | Probable | Medium | High | L1 |
| MEM50-CPP | High | Likely | Medium | Low | L1 |
| DCLO3-CPP | Low | Unlikely | High | Low | L3 |
| ERR51-CPP | Low | Probable | Medium | Low | L3 |
| FLP30-V | Low | Probable | Medium | Low | L3 |
| FIO51-CCP | Medium | Unlikely | Medium | Medium | L3 |
| DCL30-CPP | High | Probable | High | High | L2 |
| DCL37-CPP | Medium | Probable | Medium | Medium | 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 in flight is when data is being transmitted to another location and a way we can apply this policy is by encrypting the data with a key and decrypting it when it is at it location of transmission. By apply this policy we allow for safer transfer of data that prevents attacks like man in the middle from capturing any information. |
| Encryption at flight | Encryption at rest is when we have data in a disk of server that is being stored and by applying these secure polices we can add a layer of security to our data by encrypting it with a key while it is at rest and decrypting it when we need information. Allowing the right users to be able to access information and keeping unwanted users from leaking out information because it is encrypted. |
| Encryption in use | Encryption in use is compromising data in use that enables access to encrypted data at rest and in flight. We can apply the policy of more secure by only giving the right users the encrypted key and keeping those in safe places where only the right people have access to them. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a system understanding who is using the information and making sure the user is who they say they are. The policy can be applied by writing code that check data related to the user input, making sure the user is who they say they are and going even further by applying two-factor authentication to each of the users. |
| Authorization | Authorization check what the user is allowed once he or she is authenticated into the system and we can apply the policy by allowing certain user either read write or read only access as a few examples. We can also apply authorization policy by allowing only to access information in an array that belongs to only them and keeping them out of unauthorized directories. |
| Accounting | Accounting monitors the resources the user consumes during the access of the server/network and we can apply the policy by encrypting information that prevents a user from accessing without a key. While a user has access to the system we can monitor the usage of data that is used during the users access. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

## Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

# Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

# Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

# Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

# Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

# Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

# Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 20.10 | 04/18/2021 | Fully Checked | Chang | NIST |
| 7.07 | 04/18/2021 | Implement Checked | Chang | SEI CERT C |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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