**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 | Input validation is a process that ensures data entered by users is safe and reliable. This means examining and verifying the integrity, formatting, and type of input data to protect against errors, vulnerabilities, and malicious activities. This will aid with preventing/defending against data corruption, injection attacks, buffer overflows, etc. |
| 1. Heed Compiler Warnings | To heed compiler warnings is to pay attention to warnings that are issued by the compiler during the code compilation process. Compilers will provide warnings that alert developers about potential issues or coding that could lead to bugs, errors, and more. |
| 1. Architect and Design for Security Policies | The code that is created should abide by, implement, and enforce security policies. This section makes sure that the code meets requirements in many areas, for example if creating code that requires different privileges, making sure that there are subsystems that have the matching privileges. |
| 1. Keep It Simple | To keep it simple is to make sure that your code is readable, and makes sense. If the code is too complex, then it could be easy to miss errors when the code is being ran. This is avoided by creating smaller, more readable code. |
| 1. Default Deny | Default deny is to deny all access in a code by default and create special accesses/permissions for performing other actions. This means access to most (if not all) resources should be restricted unless permission is explicitly granted. |
| 1. Adhere to the Principle of Least Privilege | This ties into the Default Deny section, because it means that processes should execute with the least amount of privilege necessary to complete the job. This helps minimize potential impacts of security breaches/compromised accounts. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data sent to other systems is to properly clean and validate data before transmitting it to other systems. It mitigates potential vulnerabilities and attacks triggered by malformed/malicious data. This ensures the integrity and reliability of the software system. |
| 1. Practice Defense in Depth | Practicing defense in depth is crucial because it ensures that if a defense method fails, there are multiple additional layers of defense available to protect a system. This makes it more difficult for attackers to get through and minimizes any potential damage caused by breaches. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance exists to help find and eliminate vulnerabilities within the code. This could be done in many ways, such as through a contracted company paid to try and hack into your code and find vulnerabilities, or through multiple levels of testing. |
| 1. Adopt a Secure Coding Standard | To adopt a secure coding standard means to just check whatever the secure coding standard is for whichever language you are working with. |

### 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** | **Data Type Coding Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Ensure that any operations on signed integers do not result in overflow. |

| **Noncompliant Code** |
| --- |
| An unsigned integer here will result in overflow |
| void func(signed int a, signed int b){  signed int sum = a + b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This solution prevents the operation from overflowing and deals with overflow. |
| #include <limits.h>  void func(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):** There’s a possibility that an unsigned integer will result in overflow, which could then be taken advantage of by an attacker. We need to validate input data in order to prevent out of bounds memory errors. If we pay attention to compiler warnings, we are able to prevent errors from happening. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2023.1 | CERT\_C-1NT32-a  CERT\_C-INT32-b  CERT\_C-INT32-c | Ensure operations on signed integers do not result in overflow |
| Helix QAC | 2022.4 | C2800, C2801, C2802, C2803, C2860, C2861, C2862, C2863 | Implemented |
| TrustInSoft Analyzer | 1.44 | signed\_overflow | Ensure operations on signed integers do not result in overflow. |
| Coverity | 2022.12.2 | BAD\_SHIFT TAINTED\_SCALER | Ensure operations on signed integers do not result in overflow. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Use valid range of integer data types to prevent integer overflow between both signed and unsigned integer conversions |

| **Noncompliant Code** |
| --- |
| Converting data can lead to values that fall outside the acceptable range and may result in the truncation of the outcome |
| unsigned long int abc = ULONG\_MAX;  signed char def;  def = (signed char)abc; |

| **Compliant Code** |
| --- |
| Result does not exceed the max |
| unsigned long int abc = ULONG\_MAX;  signed char def;  if (abc <= SCHAR\_MAX){  def = (signed char)abc;  }  else {  /\* error \*/  } |

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

| **Principles(s):** Validate input data in order to catch any possible attacks. This will work as effective quality assurance, mitigating the problem. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2023a | CERT C: Rule INT30-C | Checks for unsigned integer overflow and unsigned integer constant overflow. |
| Coverity | 2022.12.2 | INTEGER\_OVERFLOW | Ensure operations on signed integers do not result in overflow. |
| CodeSonar | 7.3 | 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 | Avoid overflow of allocation size. Unreasonable size argument. |
| Parasoft C/C++ test | 2023.1 | CERT\_C-1NT32-a  CERT\_C-INT32-b  CERT\_C-INT32-c | Ensure operations on signed integers do not result in overflow. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Ensure that the allocated storage for strings is adequately sized to accommodate the character data along with the null terminator. |

| **Noncompliant Code** |
| --- |
| Can produce buffer overflow due to input being unbounded |
| #include <iostream>  void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| We use std::string to prevent 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):** Watch for warnings of overflow, prevent string attacks by sanitizing data. Ensure you are validating input. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3 | LANG.MEM.BO  LANG.MEM.TO  MISC.MEM.NTERM  BADFUNC.BO.\* | Buffer overrun. Type overrun. No space for null terminator. |
| Coverity | 2022.12.2 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| Helix QAC | 2022.4 | C2840, C5009, C5038  C++0145, C++5009, C++5038 | N/A |
| Parasoft C/C++ test | 2023.1 | CERT\_C-STR31-a  CERT\_C-STR31-b  CERT\_C-STR31-c  CERT\_C-STR31-d  CERT\_C-STR31-e | Avoid accessing arrays out of  bounds.  Avoid overflow when writing to a  buffer.  Prevent buffer overflows from  tainted data. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Cleanse the data that is passed to intricate subsystems. It is important to note that string data being passed to these subsystems might include special characters that have the potential to execute commands or trigger unintended actions. |

| **Noncompliant Code** |
| --- |
| Inputs email address to a buffer and then uses string as argument to call system |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| Data is cleaned and validated before being passed to the string |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  "1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {  \*cp = '\_';  } |

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

| **Principles(s):** As before, validate input, watch for compiler warnings, and don’t over complicate the code in order to avoid room for making more errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | dangling\_pointer\_use | N/A |
| Helix QAC | 2022.4 | DF4721, DF4722, DF4723 | N/A |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-MEM56-a | Do not store an already owned  pointer value in an unrelated  smart pointer |
| Polyspace Bug Finder | R2023a | CERT C++: MEM56-CPP | Checks for use of already owned  pointers (rule fully covered) |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not read uninitialized memory |

| **Noncompliant Code** |
| --- |
| I is uninitialized so will act undefined |
| #include <iostream>  void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| I is now initialized |
| #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):** Make sure to ensure consistency and employ secure coding standards. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3 | LANG.STRUCT.RPL LANG.MEM.UVAR | Return pointer to local uninitialized variable. |
| HelixQAC | 2022.4 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 | N/A |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| Polyspace Bug Finder | R2023a | CERT C++: EXP53-CPP | Checks for non initialized variable and non initialized pointer. Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use assert statements to handle errors that tst for conditions that should never be true |

| **Noncompliant Code** |
| --- |
| Relies only on assertion to handle error |
| myError = thisIsWrong(a, b);  /\* no code to handle errors \*/  ASSERT(!myError); |

| **Compliant Code** |
| --- |
| Error is reset to zero before assertion is reached |
| myError = thisIsRight(a, b);  /\* code to handle errors and reset myError \*/  ASSERT(!myError); |

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

| **Principles(s):** Ensure developer team is trained in secure memory allocation techniques, consistency in code, and performs routine quality control checks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.3 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Coverity | 2022.12.2 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect. |
| Parasoft C/C++ test | 2023.1 | CERT\_C-MSC11-a | Incorporate diagnostic tests using assertions |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| Both functions don’t catch any exceptions thrown |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  f();  } |

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

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

| **Principles(s):** Ensure developer team is trained in secure memory allocation techniques, consistency in code, and performs routine quality control checks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | potentially-throwing-staticinitialization | Partially checked. |
| CodeSonar | 7.3 | LANG.STRUCT.EXCP.THROW | Use of throw. |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-ERR58-a | Exceptions shall be raised only after start-up and before termination of the program. |
| Polyspace Bug Finder | R2023a | CERT C++: ERR58-CPP | Checks for exceptions raised during program startup |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Function return values | [STD-008-CPP] | Functions that return a value must return a value from all exits |

| **Noncompliant Code** |
| --- |
| Will not return value if a is not less than zero |
| int abs(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| Able to return value |
| int abs(int a) {  if (a < 0) {  return -a;  }  return a;  } |

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

| **Principles(s):** Watch for compiler warnings, and use quality control to catch errors as soon as possible. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | Return-implicit | Fully checked. |
| CodeSonar | 7.3 | LANG.STRUCT.MRS | Missing return statement. |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-MSC52-a | All exit paths from a function, except main(), with non-void return type shall have an explicit return statement with an expression. |
| Polyspace Bug Finder | R2023a | CERT C++: MSC52-CPP | Checks for missing return statements (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Commenting | [STD-009-CPP] | Commenting the code is common practice and lets other developers understand what that block of code is meant to do. Be consistent in your style. |

| **Noncompliant Code** |
| --- |
| No comments means that it takes longer for someone to understand what your code is trying to accomplish |
| **function** sourceCodeComment () {  **var** comment = document.getElementbyID("Code Comment").value;  **if** (comment != **null** && comment != '') {  **return** console.log("Thank you for your comment.")  } |

| **Compliant Code** |
| --- |
| Properly commented code makes it clear what the goal is of this code block |
| **//Checks if a comment exists, if so then returns a thank you**  **function** sourceCodeComment () {  **var** comment = document.getElementbyID("Code Comment").value;  **if** (comment != **null** && comment != '') {  **return** console.log("Thank you for your comment.")  } |

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

| **Principles(s):** Consistancy, adopt a secure coding standard, make sure developers understand when to use comments, and how to write them efficiently. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ECLAIR | 1.2 | CC2.MSC04 | Fully implemented |
| Parasoft C/C++ test | 2023.1 | CERT\_C-MSC04-a  CERT\_C-MSC04-b  CERT\_C-MSC04-c  CERT\_C-MSC04-d | The character sequence /\* or // shall not be used within a C-style comment The character sequence /\* shall not be used within a C++-style comment Line-splicing shall not be used in // comments |
| Polyspace Bug Finder | R2023a | CERT C: Rec. MSC04-C | Checks for use of /\* and // within a comment (rule partially covered) |
| RuleChecker | 23.04 | mmline-comment sline-comment sline-splicing smline-comment | Partially checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Qualifying reference types | [STD-010-CPP] | Never qualify a reference type with const or volatile. |

| **Noncompliant Code** |
| --- |
| a const-qualified reference to a char is formed instead of a reference to a const-qualified char. This results in undefined behavior. |
| #include <iostream>    **void** f(**char** c) {  **const** **char** &p = c;    p = 'p'; // Error: read-only variable is not assignable    std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| This compliant solution removes the const qualifier. |
| #include <iostream>  void f(char c) {  char &p = c;  p = 'p';  std::cout << c << std::endl;  } |

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

| **Principles(s):** Build code to prevent vulnerabilities, keep it simple to avoid overcomplication causing errors, make tests that are as effective as possible |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | Return-implicit | Fully checked |
| Axivion | 7.5 | CertC++MSC52 | N/A |
| Parasoft C/C++ test | 2023.1 | CERT\_CPP-MSC52-a | All exit paths form a function with non void return type shall have an explicit return statement with an expression |
| Polyspace Bag Finder | R2023a | Cert C++: MSC52-a | Checks for missing return statements |

### 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 | P9 | 2 |
| STD-002-CPP | High | Likely | High | P9 | 2 |
| STD-003-CPP | High | Likely | Medium | P18 | 1 |
| STD-004-CPP | High | Likely | Medium | P18 | 1 |
| STD-005-CPP | High | Probable | Medium | P12 | 1 |
| STD-006-CPP | Low | Unlikely | High | P1 | 3 |
| STD-007-CPP | Low | Likely | Low | P9 | 2 |
| STD-008-CPP | Medium | Probable | Medium | P8 | 2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | 3 |
| STD-010-CPP | Medium | Probable | Medium | P8 | 2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption in rest helps protect data that is stored on a disk. It applies because we need to keep that data secure from possible attacks should the hardware get into the hands of others, or should an attacker get into your personal system. |
| Encryption at flight | Encryption at flight protects data confidentiality between a client and a server. This data can be intercepted by an attacker, so it protects it through encryption. |
| Encryption in use | Encryption in use ensures that there is continuous protection of data no matter what stage or location. This prevents it from being unsecured. This policy is important because it makes sure data is always being safeguarded, reducing the risk of a possible breach. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is used to verify the identity of a user. This can be done in a few ways, such as usernames, passwords, biometrics, etc. Through authentication, systems are ensuring that only authorized users have access to resources. This keeps the information secure. |
| Authorization | Authorization determines what roles, privileges, and accesses users have after authentication. This controls access to specific functions, resources, and data in a asystem, ensuring that only authorized users are performing certain actions. This keeps information secure because people can only access data that they are set to be able to access. |
| Accounting | Accounting tracks the use of resources and anctions done by users that have been authenticated and authorized. Examples of information logged are time of access, what they did, and who did it. This helps to recognize who might be accessing things that they do not need to 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 |  |
| 2.0 | 05/28/2023 | Added oding standards | Alexis Fuerte |  |
| 3.0 | 06/11/2023 | Updated all polcies | Alexis Fuerte |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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