**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 used to detect unauthorized input before it is processed by the application. This should be done as soon as data is received from the external party to ensure properly formed data is entering the workflow. Inputs from all external sources should always be validated to help prevent Input Validation Attacks and SQL Injections with a focus on syntactic and semantic levels. |
| 1. Heed Compiler Warnings | Code should be compiled using the highest warning level available. Use static and dynamic analysis tools to detect any additional security flaws. Always modify the code to eliminate these warnings and flaws. |
| 1. Architect and Design for Security Policies | When creating a software architecture, design the software to implement and enforce security policies. When designing a system that requires different privileges at different times, divide the system into subsystems with appropriate privileges. |
| 1. Keep It Simple | Complex designs are more likely to have errors in their implementation, configuration, and use. Keeping the design small and simple will help prevent these errors. |
| 1. Default Deny | Access decisions should be based on permission instead of exclusion. By default, access should be denied with specific conditions to permit access. |
| 1. Adhere to the Principle of Least Privilege | Processes should execute with the least amount of privilege necessary to complete the job. If elevated privileges are required, they should be used for as little time as possible. This will help to reduce the opportunity an attacker has to execute arbitrary code with elevated privileges. |
| 1. Sanitize Data Sent to Other Systems | Data passed through complex subsystems should be sanitized before invoking the subsystem. This helps to ensure attackers cannot invoke unused functionality through injection attacks. |
| 1. Practice Defense in Depth | Defense in Depth helps prevent attacks by having multiple layers the attacker must go through to penetrate the application. If one layer turns out to be inadequate, there are other layers underneath to help prevent the attack from occurring. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques can help identify and eliminate vulnerabilities. External security reviews, independent security reviews, multiple types of testing, and multiple testing phases can create a more secure system. |
| 1. Adopt a Secure Coding Standard | No matter what language or platform you are using, there needs to be a secure coding standard that is followed. |

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Ensure that operations on signed integers do not result in overflow.  Unsigned integers should be used on values that cannot become negative and signed integers on values that can become negative. |

| **Noncompliant Code** |
| --- |
| The code does not consider that the unsigned integer will cause an infinite loop. |
| char a[MAX\_ARRAY\_SIZE] = /\*initialize value\*/;  size\_t count = /\*initialize value\*/;  for (unsigned int i = count – 2; i >= 0; i--) {  a[i] += a[i+1];  } |

| **Compliant Code** |
| --- |
| Since size\_t is an unsigned type, this will not create an infinite loop. |
| char a[MAX\_ARRAY\_SIZE] = /\*initialize value\*/;  size\_t count = /\*initialize value\*/;  for (size\_t i = count – 2; i != MAX\_ARRAY\_SIZE; i--) {  a[i] += a[i+1];  } |

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

| **Principles(s):** Validate input data, heed complier warnings, and effective quality assurance techniques.  We can prevent out of bounds memory errors by properly validating input data, by heeding compiler warnings we can catch errors before production and using effective quality assurance techniques can help catch these errors using analysis. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2022.2 | CERT\_C-INT32-a  CERT\_C-INT32-b  CERT\_C-INT32-c | Ensure that operations on signed integers do not result in overflow. |
| Helix QAC | 2021.2 | C2800, C2801, C2802, C2803, C2860, C2861, C2862, C2863 | Implemented |
| TrustInSoft Analyzer | 1.38 | signed\_overflow | Ensure operations on signed integers do not result in overflow. |
| Coverity | 2017.07 | BAD\_SHIFT  TAINTED\_SCALAR | 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 ranging of integer data types to prevent integer overflow between signed and unsigned integer conversions. |

| **Noncompliant Code** |
| --- |
| Conversion can result in out of range values and can cause truncation of the result. |
| unsigned long int abc = ULONG\_MAX;  signed char def;  def = (signed char)abc; |

| **Compliant Code** |
| --- |
| Ensure the result will 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):** Use effective quality assurance techniques to mitigate these problems through analysis tools. Validate input data to catch these mistakes. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2022b | CERT C: Rule INT30-C | Checks for unsigned integer overflow and unsigned integer constant overflow. |
| Coverity | 2017.07 | INTEGER\_OVERFLOW | Ensure operations on signed integers do not result in overflow. |
| CodeSonar | 7.2p0 | 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 | 2022.2 | CERT\_C-INT32-a  CERT\_C-INT32-b  CERT\_C-INT32-c | Ensure that operations on signed integers do not result in overflow. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Buffer boundaries not set and incorrect string sizes can lead to runtime errors and buffer overflows. |

| **Noncompliant Code** |
| --- |
| An input of more than 11 characters will result in out of bounds. |
| #include <iostream>  int main(void) {  char buffer[12];  std::cin >> buffer;  std::cout << buffer << endl;  } |

| **Compliant Code** |
| --- |
| Setting the width to the size of the array buffer eliminates overflow. |
| #include <iostream>  int main(void) {  char buffer[12];  std::cin.width(12);  std::cin >> buffer;  std::cout << buffer << endl;  }  string s;  string::iterator i;  for (i = s.begin(); i != s.end(); ++i) {  cout << \*i;  } |

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

| **Principles(s):** Heed compiler warnings of overflow, sanitize data to prevent string attacks, and validate input data with appropriate string function. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.2p0 | LANG.MEM.BO  LANG.MEM.TO  MISC.MEM.NTERM  BADFUNC.BO.\* | Buffer overrun.  Type overrun.  No space for null terminator. |
| Coverity | 2017.07 | 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 | 2022.2 | 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] | Already owned pointer values should not be stored in an unrelated smart pointer. |

| **Noncompliant Code** |
| --- |
| Two unrelated smart pointers are constructed from the same pointer value. When p2 is destroyed it deletes the same point value as p1. |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| Calls std::make\_shared() instead of allocating a raw pointer and storing the value in a local variable. |
| #include <memory>  void f() {  std::shared\_ptr<int> p1 = std::make\_shared<int>();  std::shared\_ptr<int> p2(p1);  } |

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

| **Principles(s):** Validate input, heed compiler warnings to avoid errors, and keep it simple to avoid making mistakes that are not necessary. |
| --- |

**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 | 2022.4 | CERT\_CPP-MEM56-a | Do not store an already owned pointer value in an unrelated smart pointer. |
| Polyspace Bug Finder | R2022b | 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** |
| --- |
| Local variable i is uninitialized which will result in undefined behavior. |
| #include <iostream>  void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| The variable needs to be initialized before printing. |
| #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):** Heed compiler warnings to prevent this simple mistake, use effective quality assurance techniques to catch this mistake early, and adopt a secure coding standard to ensure consistency. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.2p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local uninitialized variable. |
| Helix QAC | 2022.4 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 | N/A |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| Polyspace Bug Finder | R2022b | 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 test for conditions that should never be true. |

| **Noncompliant Code** |
| --- |
| Do not rely only on the assertion to handle the error condition. |
| myError = thisIsWrong(a, b);  /\* no code to handle errors \*/  ASSERT(!myError); |

| **Compliant Code** |
| --- |
| If the code works properly, the error should be handled and reset to zero before the 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):** Architect and design for security policies makes sure the team is trained in secure memory allocation techniques, adopt a secure coding standard to ensure consistency, and use quality assurance techniques to test for these vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.2p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Coverity | 2017.07 | 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 | 2022.2 | CERT\_C-MSC11-a | Incorporate diagnostic tests using assertions. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle exceptions thrown before main() begins. |

| **Noncompliant Code** |
| --- |
| S may throw an exception that is not caught when globalS is constructed. |
| Struct S {  S() noexcept(false);  };  static S globalS; |

| **Compliant Code** |
| --- |
| A try catch block will help catch exception thrown. |
| Struct S {  S() noexcept(false);  };  S &globalS() {  try {  static S s;  return s;  }  catch() {  /\* code to handle error \*/  } |

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

| **Principles(s):** Adopt a secure coding standard to ensure consistency, use quality assurance techniques to run assertions on each scan, and train the team using architect and design for security to help with consistency. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | potentially-throwing-static-initialization | Partially checked. |
| CodeSonar | 7.2p0 | LANG.STRUCT.EXCP.THROW | Use of throw. |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-ERR58-a | Exceptions shall be raised only after start-up and before termination of the program. |
| Polyspace Bug Finder | R2022b | CERT C++: ERR58-CPP | Checks for exceptions raised during program startup (rule fully covered) |

#### Coding Standard 8

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

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

| **Compliant Code** |
| --- |
| Code will return a value for either situations. |
| 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):** Heed compiler warnings to ensure all possible mistakes are fixed and use effective quality assurance techniques in order to catch simple errors as early as possible. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | return-implicit | Fully checked. |
| CodeSonar | 7.2p0 | LANG.STRUCT.MRS | Missing return statement. |
| Parasoft C/C++ test | 2022.2 | 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 | R2022b | CERT C++: MSC52-CPP | Checks for missing return statements (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Comments | [STD-009-CPP] | All code needs to be commented in order to tell the next person what you are doing. Use comments consistently and in a readable fashion. |

| **Noncompliant Code** |
| --- |
| Does not have any comments, could be unclear for next person. |
| if (std::find(v.begin(), v.end(), element) != v.end()) {  Process(element);  } |

| **Compliant Code** |
| --- |
| Tells the next person what this is used for. |
| /\* Process "element" unless it was already processed \*/  if (std::find(v.begin(), v.end(), element) != v.end()) {  Process(element);  } |

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

| **Principles(s):** Using architect and design for security policies and adopt a secure coding standard ensure the team is trained to the highest level and they understand when and how to use comments. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ÉCLAIR | 1.2 | CC2.MSC04 | Fully implemented. |
| Parasoft C/C++ test | 2022.2 | 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 | R2022b | CERT C: Rec. MSC04-C | Checks for use of /\* and // within a comment (rule partially covered) |
| RuleChecker | 22.04 | mmline-comment  sline-comment  sline-splicing  smline-comment | Partially checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-010-CPP] | Do not delete polymorphic object without virtual destructor. |

| **Noncompliant Code** |
| --- |
| Implicitly declared destructor is not declared as virtual; when b is deleted it results in undefined behavior because Base does not have a virtual destructor. |
| struct Base {  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

| **Compliant Code** |
| --- |
| The destructor for Base has an explicitly declared virtual destructor, ensuring the polymorphic delete operation results in well-defined behavior. |
| struct Base {  virtual ~Base() = default;  virtual void f();  };    struct Derived : Base {};    void f() {  Base \*b = new Derived();  // ...  delete b;  } |

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

| **Principles(s):** Heed compiler warnings notifying you of an error and use effective quality assurance techniques to ensure errors are caught as early as possible. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.2p0 | LANG.STRUCT.DNVD | Delete with non-virtual destructor |
| Parasoft C/C++ test | 2022.2 | CERT\_CPP-OOP52-a | Define a virtual destructor in classes used as base classes which have virtual functions. |
| Polyspace Bug Finder | R2022b | CERT C++: OOP52-CPP | Checks for situations when a class has virtual functions but not a virtual destructor (rule partially covered) |
| RuleChecker | 22.10 | Non-virtual-public-destructor-in-non-final-class | Partially checked. |

### 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 | Likely | High | P9 | L2 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Probable | Medium | P12 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | Low | Likely | Low | P9 | L2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption that is used to help protect data that is stored on a disk. This policy applies because sensitive data may get stored in systems we build, and it is important to keep that data secure from any attacks. |
| Encryption at flight | Encryption of data that moves over a network. This applies because as data is sent and requested between servers and clients, encryption at flight protects this data if gets intercepted by an attacker. |
| Encryption in use | Ensures sensitive data is never left unsecured, regardless of lifecycle stage or location. This policy applies because data needs to be always secured, especially when it is in use. This would apply to a user entering a password or any sensitive information. This is done through hashing and user authentication just to name a couple. Ensuring data is always secured greatly reduces the risk of a breach. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of identifying a user through valid credentials, such as username, password, phone number, keys, and biometrics. By using authentication, data is kept secure, and users can only access information they are authorized to access. Authentication should be performed on every new user account created regardless of privileges. |
| Authorization | Authorization is the right to access data based on the user’s role and privilege. Changes to the database should only be performed by authorized individuals. By adhering to the principle of least privilege, users can only access the least set of privileges necessary for them to complete the job, and any elevated permission should only be granted for a minimum time. |
| Accounting | Accounting is the process of keeping track of a user’s activity while accessing the network resources. This includes the amount of time the user spends in the network, and files accessed while there, and the amount of data transferred during the session. |

**\***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 |  |
| 1.5 | 01/22/2023 | Coding Standards | Breanna Hodges |  |
| 2.0 | 02/17/2023 | Updated All Policies | Breanna Hodges |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

https://safeonline.ng/web-developers/secure-coding-practices/