**Green Pace Developer: Security Policy Guide**



# 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 | Where you validate input from all untrusted data sources. This will reduce or eliminate key software vulnerabilities. When validating, awareness is key when involved with network interfaces, environmental variables, and external data sources. |
| 1. Heed Compiler Warnings | To eliminate compiler warnings, compile code with the highest warning level settings, and modify code when warnings are received. To identify and eliminate security flaws, use static and dynamic analysis tools. |
| 1. Architect and Design for Security Policies | Plan and design your software to re-enforce security policies. This could require dual access requirements, different levels of security access based on needs, and pre-set protocols in place to deal with any variations that deviate from established policies. |
| 1. Keep It Simple | Keep coding and design requirements simple to avoid a complex design that could increase errors in every aspect of the software lifecycle, such as the implementation, or configuration stages. The more security principles you have in place will require a more complex strategy. |
| 1. Default Deny | This implies that by default, access is denied, and protocols are set in place that identify conditions for granted access or increased permissions. |
| 1. Adhere to the Principle of Least Privilege | This can be defined as the lowest level of access necessary to complete the mission set. If an increase permission is necessary, it would be on a temporary basis only, until its need is met, and then access is then reduced. This can limit access by outside as well as inside threats and reduce their opportunities to inflict damage. |
| 1. Sanitize Data Sent to Other Systems | Sanitize or clean data so that the context for understanding data sent is only understood by the calling process. Other systems would include complex subsystems such as cmd shells and relational databases. This will eliminate the hackers accessing unused functionality via SQL or other injection attacks. |
| 1. Practice Defense in Depth | Create multi-level overlapping defensive strategies. If one layer of security is compromised, other overlapping layers will kick in and prevent a breach in security or exploitable vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques will identify and eliminate vulnerabilities through penetration testing, source code audits and other testing phases. As well, independent security reviews bring an independent outside perspective and help in identifying issues you might not have been aware of. |
| 1. Adopt a Secure Coding Standard | Whatever language or platform you are using, develop and apply a secure coding standard. The more robust and thought out it is, the more efficient and effective it will become. |

### 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] | Allow only enumeration values within specified parameters |

| **Noncompliant Code** |
| --- |
| Code will check compliancy, and project handle errors if the value is outside of the accepted parameters |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  if (enumVar < First || enumVar > Third) {  // Handle error  }  } |
|  |

| **Compliant Code** |
| --- |
| Code will check compliancy by checking the value before doing the conversion to ensure the value is acceptable. |
| enum EnumType {  First,  Second,  Third  };  void f(int intVar) {  if (enumVar < First || enumVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** If enumeration values do not fall within specified parameters, a buffer overflow can occur, which opens up a vulnerability for outside hackers. To avoid this, data integrity needs to be enforced with arithmetic validation. Most probable outcome is a mathematical error causing code failure. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 |  |
| CodeSonar | 7.0p0 | LANG.CAST.COERCE  LANG.CAST.VALUE | Coercion alters value  Cast alters value |
| Helix QAC | 2022.1 | C++3013 |  |
| Parasoft C/C++ test | 2021.2 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Define identifiers and valid references and pointers correctly to reduce likelihood of identifiers not defining |

| **Noncompliant Code** |
| --- |
| Naming standards incorrect resulting in undefined behavior |
| #ifndef \_ HEADER\_G\_  #define \_ HEADER\_G\_  // Contents of < header.g>  #endif // \_ HEADER\_G\_ |

| **Compliant Code** |
| --- |
| Removing leading and trailing underscores allowed definitions to be defined |
| #ifndef HEADER\_G  #define HEADER\_G  // Contents of < header.g>  #endif // HEADER\_G |

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

| **Principles(s):**  Heed compiler warnings – always pay attention to warnings as they are there for a reason  Architect and Design for Security Policies – building code to prevent vulnerabilities  Keep it simple – always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques – making tests that are as effective as possible  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-INT50 | - |
| [Astree](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | Reserved-identifier | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | Wreserved-id-macro  Wuser-defined-literals | -The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Never qualify a reference type with and const as it will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| A const qualified reference to char is formed. |
| #include <iostream>  void f(char b) {  char &const p = b;  p = 'p';  std::cout << b << std::endl;  } |

| **Compliant Code** |
| --- |
| Remove the const qualifier |
| #include <iostream>  void f(char b) {  char &p = b;  p = 'p';  std::cout << b << std::endl; |

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

| **Principles(s):**  Heed compiler warnings – always pay attention to warnings as they are there for a reason  Architect and Design for Security Policies – building code to prevent vulnerabilities  Keep it simple – always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques – making tests that are as effective as possible  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | CertC++-DCL52 | - |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | CERT\_CPP-DCL52-a | Never qualify a reference type with 'const' or 'volatile' |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | 0014 | - |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | Cert C++; DCL52-cpp | Checks for:  -const-qualified reference types  -Modification of const-qualified reference types  -Rule fully covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Do not store already owned pointer value in an unrelated smart pointer |

| **Noncompliant Code** |
| --- |
| Two smart pointers are created based on the same pointer value. If the variable p2 is eliminated, it deletes its pointer value. If then variable p1 is eliminated, it deletes the same pointer value, causing an error |
| #include <memory>  void f() {  int \*i = new int;  std::shared\_ptr<int> p1(i);  std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| The shared pointers are now connected via copy construction, where a new object is created form an existing object as a copy of the existing object. In this case, if one of the variables is eliminated, the value still exists until the last copy of the variable is eliminated |
| #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):** Storing an already owned pointer value in an unrelated smart pointer can result in undefined behavior, leading to vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | CERT\_CPP-MEM56-a | DO NOT store an already owned pointer value in an unrelated smart pointer |
| Astree | 20.10 | Dangling\_pointer\_use | - |
| Helix QAC | 2021.1 | - | - |
| PVS-Studio | 7.01 | V1006 | - |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Properly De-allocate allocated resources |

| **Noncompliant Code** |
| --- |
| Undefined behavior resulting from ::operator delete() attempting to free memory not returned by ::operator new() |
| #include <iostream>    struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;  // ...  delete s1;  } |

| **Compliant Code** |
| --- |
| By removing the call to ::operator delete() this will correct the issue. |
| #include <iostream>  struct S {  S() { std::cout << "S::S()" << std::endl; }  ~S() { std::cout << "S::~S()" << std::endl; }  };  void f() {  alignas(struct S) char space[sizeof(struct S)];  S \*s1 = new (&space) S;  s1->~S();  } |

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

| **Principles(s):** Passing a pointer value to a deallocation function that was not previously obtained by the matching allocation function results in undefined behavior, which can lead to exploitable vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Invalid\_dynamic\_memory\_allocation\_dangling\_pointer\_use | - |
| Helix QAC | 2021.1 | - | - |
| PVS-Studio | 7.07 | V515, V554, V611, V701, V748, V773 | - |
| PRQA QA-C++ | 4.4 | 2110, 2111, 2112, 2113, 2118, 3337, 3339, 4262, 4263, 4264 | - |
| Polyspace Bug Finder | R2020a | CERT C++: MEM51-CPP | Checks for:  -Invalid deletion of pointer  -Invalid free of pointer  -Deallocation of previously deallocated pointer  -Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use static assertion to test value of a constant expression |

| **Noncompliant Code** |
| --- |
| This uses the assert() macro to assert a property |
| #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** |
| --- |
| As this assertion involves a constant expression, a conditional statement is used |
| 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):** static assertion is a good diagnostic tool to locate software defects to limit vulnerabilities when compiling. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CERTC-DCL03 | - |
| LDRA tool suite | 9.7.1 | 44 S | Fully Implemented |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| ÉCLAIR | 1.2 | CC2.DCL03 | Fully Implemented |

#### Coding Standard 7

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

| **Noncompliant Code** |
| --- |
| The constructor for S may throw an exception which will not be caught as globalS is constructed later on |
| struct S {  S() noexcept(false);  };  static S globalS; |

| **Compliant Code** |
| --- |
| GlobalS is now a local variable and will now catch any exceptions thrown during object construction, because the constructor for S executes when function globalS is called. |
| struct S {  S() noexcept(false);  };    S &globalS() {  try {  static S s;  return s;  } catch (...) {  // Handle error, perhaps by logging it and gracefully terminating the application.  }  // Unreachable.  } |

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

| **Principles(s):** throwing an exception that cannot be caught will result in program exit and open up vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CERTC++-ERR58 | - |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-ERR58a | Exceptions raised only after start up and before program termination |
| Rule Checker | 20.10 | Potentially-throwing-static-initialization | Partially checked |
| Clang | 3.9 | CERT-ERR58-CPP | Checked by clang-tidy |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Output Input | [STD-008-CPP] | Have a positioning call when alternating output and input from a file stream |

| **Noncompliant Code** |
| --- |
| This appends data to the end of a file then reads from the same file. However, it does not include a positioning call between input and output calls, so creates an error |
| #include <fstream>  #include <string>  void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }    file << "Output data";  std::string str;  file >> str;  } |

| **Compliant Code** |
| --- |
| Here, the seekg() fuction is called between input and output to eliminate the undefined behavior. |
| #include <fstream>  #include <string>  void f(const std::string &fileName) {  std::fstream file(fileName);  if (!file.is\_open()) {  // Handle error  return;  }  file << "Output data";  std::string str;  file.seekg(0, std::ios::beg);  file >> str;  } |

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

| **Principles(s):** you will create undefined behavior by alternately inputting and outputting without a flush or positioning call |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2020a | ECRT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call  Rule fully covered |
| Helix QAC | 2021.1 | - | - |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP\_FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Namespace modifications | [STD-009-CPP] | Do not modify standard namespaces. New declarations in the namespace can cause undefined behavior |

| **Noncompliant Code** |
| --- |
| Y is added to the namespace causing undefined behavior |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| By Changing the namespace to non-standard you are placing without a reserved name |
| namespace nonstd {  int x;  } |

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

| **Principles(s):**  Architect and Design for Security Policies – building code to prevent vulnerabilities  Keep it simple – always applies as keeping code as lightweight as possible is best practice  Use Effective Quality Assurance Techniques – making tests that are as effective as possible  Adopt a secure coding standard – making security a priority helps prevent vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | CertC++: DCL58-CPP | Checks for modification of standard namespaces  Rule fully covered |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | CertC++-DCL58 | - |
| [Parasoft C/C++ test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | CERT\_CPP-DCL58-a | Do not modify the standard namespaces 'std' and 'posix |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | 4032, 4035, 4631 | - |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Value inputs | [STD-010-CPP] | Value returning functions must return a value from all code paths |

| **Noncompliant Code** |
| --- |
| Code does not include an input value for a positive input, so not all code paths return a value |
| int absolute\_value(int a) {  if (a < 0) {  return -a;  }  } |

| **Compliant Code** |
| --- |
| This code includes negative and positive input, so all code paths return a value |
| int absolute\_value(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):** Not returning a value in a value-returning function results in undefined behavior, which will result in integrity violations |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Return-implicit | Fully checked |
| Axivion Bauhaus | 6.9.0 | CertC++-MSC52 | - |
| LDRA tool suite | 9.7.1 | 2D, 36S | Fully implemented |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-MSC52a | All exit paths from a function with non-void return type shall have an explicit return statement with an expression |

### 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



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.

This automation structure for the DevOps process is sound, as it is integrated within each segment of the process. As with other products, this process continues with automating its updates and modifications after product release, leaving open new version pushes at later dates.

### 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 | rtyg | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-003-CPP | Low | Unlikely | Low | P3 | L3 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | Low | Likely | Medium | P6 | L2 |
| STD-009-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-0010-CPP | Medium | Probable | Medium | P8 | 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 for data at rest involves securely encoding data as it is written into storage and decrypting that data as it is pulled from storage for use. Using an encryption key when the data is written into storage protects it from unauthorized access. It should be used for all data of any level of sensitivity and would cause harm if accessed by unauthorized actors. |
| Encryption at flight | Encryption of data at-flight involves securely encoding data as it is being transmitted. How you will be transferring any data will determine how to apply this encryption. Implement secure protocols when using web browsers. When sending emails encrypt before sending and use digital signatures. |
| Encryption in use | Encryption of data in-use involves protecting data as it is utilized in memory, via password protected profiles protecting the memory of each profile |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This process used to prove who a user is, their userID, passwords, higher-level security such as secure tokens, CAC or PIN and other dual authentication. |
| Authorization | Once a user is authenticated and allowed access, they are only granted specific access to parts of that system. Authorized access to certain drives, folders, programs, and other data is allowed by the system administrators |
| Accounting | After authentication and authorization, you need to monitor and record |

**\***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.

**Operating system logs**

* 4- keep the coding and design requirements simple
* 5- by default, access is denied, and protocols are set in place that identify conditions
* 6- limit to the lowest level of access necessary to complete the mission set
* 8- Create multi-level overlapping defensive strategies
* 10- develop and apply a secure coding standard

**Firewall logs**

* 4- keep the coding and design requirements simple
* 5- by default, access is denied, and protocols are set in place that identify conditions
* 6- limit to the lowest level of access necessary to complete the mission set
* 7- prevent unnecessary data from being transmitted
* 8- Create multi-level overlapping defensive strategies
* 10- develop and apply a secure coding standard

**Anti-malware logs**

* 4- keep the coding and design requirements simple
* 5- by default, access is denied, and protocols are set in place that identify conditions
* 6- limit to the lowest level of access necessary to complete the mission set
* 7- prevent unnecessary data from being transmitted
* 8- Create multi-level overlapping defensive strategies
* 10- develop and apply a secure coding standard

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 6/11/2022 | Project One | Michael Clisbee | Michael Clisbee |
| 1.2 | 6/11/2022 | Final Revision | Michael Clisbee | Michael Clisbee |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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