**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 | Proper input validation can eliminate most software vulnerabilities |
| 1. Heed Compiler Warnings | Compile code on the highest warning level. Also use both static and dynamic analysis tools to find and eliminate security threats. |
| 1. Architect and Design for Security Policies | Design the system to implement and enforce security policies. One way to help with this is using an appropriate privilege set instead of requiring different privileges at different times. |
| 1. Keep It Simple | By keeping the program simple it decreases the likelihood of missing protection where if the system is complex it allows for more areas that can cause a breach in the system. |
| 1. Default Deny | Base access decision on permission instead of exclusion. Which means access will be denied unless the person is supposed to be there. |
| 1. Adhere to the Principle of Least Privilege | This means only give out enough permission to get the job done and no more than that. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data being sent to other systems by doing this it will lower the risk of attackers using an unused function that could break the system. |
| 1. Practice Defense in Depth | Use multiple lines of defense to make it difficult for an attacker to break the system. For instance, using a secure coding technique with a secure runtime environment. |
| 1. Use Effective Quality Assurance Techniques | Using quality assurance techniques can be effective in finding and eliminating security risks. |
| 1. Adopt a Secure Coding Standard | Use security techniques that apply to the language and platform being used. |

## 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] | Never qualify a reference type with const or volatile |

| **Noncompliant Code** |
| --- |
| This noncompliant code example correctly declares p to be a reference to a const-qualified char. The subsequent modification of p makes the program ill-formed. |
| #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** |
| --- |
| 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):** A const or volatile reference type may result in undefined behavior instead of a fatal diagnostic, causing unexpected values to be stored and leading to possible data integrity violations. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | **CertC++-DCL52** |  |
| Parasoft C/C++ test | 2020.2 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| Polyspace Bug Finder | R2020a | **0014** | Checks for:  const-qualified reference types  Modification of const-qualified reference types  Rule fully covered. |
| PRQA QA-C++ | 4.4 |  | [Insert text.] |
| Clang | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |
| SonarQube C/C++ Plugin | 4.10 | S3708 |  |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| (Bounds Checking) attempts to check whether a given value is within the range of acceptable enumeration values |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| checks that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    if (intVar < First || intVar > 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):** It is possible for unspecified values to result in a buffer overflow, leading to the execution of arbitrary code by an attacker. However, because enumerators are rarely used for indexing into arrays or other forms of pointer arithmetic, it is more likely that this scenario will result in data integrity violations rather than arbitrary code execution. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-INT50 |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PRQA QA-C++ | 4.4 | 3013 |  |
| PVS-Studio | 7.07 | V1016 |  |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Do not attempt to create a std::string from a null pointer |

| **Noncompliant Code** |
| --- |
| a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can lead to undefined behavior when the environment variable does not exist |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::**getenv**("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| the results from the call to std::getenv() are checked for null before the std::string object is constructed |
| #include <cstdlib>  #include <string>    void f() {    const **char** \*tmpPtrVal = std::**getenv**("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

|  |
| --- |
| **Principles(s):** Dereferencing a null pointer is undefined behavior, typically abnormal program termination. In some situations, however, dereferencing a null pointer can lead to the execution of arbitrary code [Jack 2007, van Sprundel 2006]. The indicated severity is for this more severe case; on platforms where it is not possible to exploit a null pointer dereference to execute arbitrary code, the actual severity is low. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | assert\_failure |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-STR51-a | Avoid null pointer dereferencing |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Use valid references, pointers and iterators to reference elements of a basic\_string |

| **Noncompliant Code** |
| --- |
| Copies input into a std::string, replacing semicolon (;) characters with spaces. |
| #include <string>    void f(const std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();    for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

| **Compliant Code** |
| --- |
| The value of the iterator loc is updated as a result of each call to insert() so that the invalidated iterator is never accessed. The updated iterator is then incremented at the end of the loop. |
| #include <string>    void f(const std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();    for (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      loc = email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

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

|  |
| --- |
| **Principles(s):** Using an invalid reference, pointer, or iterator to a string object could allow an attacker to run arbitrary code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |

### Coding Standard 5

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

| **Noncompliant Code** |
| --- |
| s is dereferenced after it has been deallocated. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| The dynamically allocated memory is not deallocated until it is no longer required. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    s->f();    delete s;  } |

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

|  |
| --- |
| **Principles(s):** Reading previously dynamically allocated memory after it has been deallocated can lead to abnormal program termination and denial-of-service attacks. Writing memory that has been deallocated can lead to the execution of arbitrary code with the permissions of the vulnerable process. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Dangling\_pointer\_use |  |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-MEM50 |  |
| Clang | 3.9 | clang-analyzer-cplusplus.NewDelete  clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| CodeSonar | 6.0p0 | ALLOC.UAF | Use after free |
| LDRA tool suite | 9.7.1 | 483 S, 484 S | Partially implemented |
| Parasoft Insure++ |  |  | Runetime detection |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Do not call a function with a mismatched language linkage |

| **Noncompliant Code** |
| --- |
| The call\_java\_fn\_ptr() function expects to receive a function pointer with "java" language linkage because that function pointer will be used by a Java interpreter to call back into the C++ code |
| extern "java" typedef void (\*java\_callback)(**int**);    extern void call\_java\_fn\_ptr(java\_callback callback);  void callback\_func(**int**);    void f() {    call\_java\_fn\_ptr(callback\_func);  } |

| **Compliant Code** |
| --- |
| The callback\_func() function is given "java" language linkage to match the language linkage for java\_callback. |
| extern "java" typedef void (\*java\_callback)(**int**);    extern void call\_java\_fn\_ptr(java\_callback callback);  extern "java" void callback\_func(**int**);    void f() {    call\_java\_fn\_ptr(callback\_func);  } |

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

|  |
| --- |
| **Principles(s):** Mismatched language linkage specifications generally do not create exploitable security vulnerabilities between the C and C++ language linkages. However, other language linkages exist where the undefined behavior is more likely to result in abnormal program execution, including exploitable vulnerabilities. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klocwork | 2018 | PORTING.CAST.PTR.FLTPNT  PORTING.CAST.PTR  PORTING.CAST.PTR.FLTPNT  PORTING.CAST.PTR.SIZE  PORTING.CAST.SIZE  MISRA.CAST.PTR.UNRELATED  MISRA.CAST.PTR\_TO\_INT |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-EXP56-a | Do not call a function with a mismatched language linkage |
| PRQA QA\_C++ | 4.4 | 3033. 3038 |  |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| The call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func() might throw an exception. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {    if (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

| **Compliant Code** |
| --- |
| f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| #include <cstdlib>    void throwing\_func() noexcept(false);    void f() { // Not invoked by the program except as an exit handler.    try {      throwing\_func();    } catch (...) {      // Handle error    }  }    **int** main() {    if (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } |

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

|  |
| --- |
| **Principles(s):** Allowing the application to abnormally terminate can lead to resources not being freed, closed, and so on. It is frequently a vector for denial-of-service attacks. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Stdlib-use | Partially checked |
| CodeSonar | 6.0p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of Abort  Use of exit |
| Klockwork | 2018 | MISRA.CATCH.ALL |  |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-008-CPP] | Close files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| A std::fstream object file is constructed. The constructor for std::fstream calls std::basic\_filebuf<T>::open(), and the default std::terminate\_handler called by std::terminate() is std::abort(), which does not call destructors. Consequently, the underlying std::basic\_filebuf<T> object maintained by the object is not properly closed |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    file.close();    if (file.fail()) {      // Handle error    }    std::terminate();  } |

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

|  |
| --- |
| **Principles(s):** Failing to properly close files may allow an attacker to exhaust system resources and can increase the risk that data written into in-memory file buffers will not be flushed in the event of abnormal program termination. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | ALLOC.LEAK | Leak |
| Klocwork | 2018 | RH.LEAK |  |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-F1051-a | Ensure Resources are freed |
| Polyspace Bug Finder | R2020a | CERT C++: FIO51-CPP | Checks for resource leak |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous | [STD-009-CPP] | Ensure random number generator is properly seeded |

| **Noncompliant Code** |
| --- |
| The example generates a sequence of 10 pseudorandom numbers using the Mersenne Twister engine. No matter how many times this code is executed, it always produces the same sequence because the default seed is used for the engine. |
| #include <random>  #include <iostream>    void f() {    std::mt19937 engine;      for (**int** i = 0; i < 10; ++i) {      std::cout << engine() << ", ";    }  } |

| **Compliant Code** |
| --- |
| uses std::random\_device to generate a random value for seeding the Mersenne Twister engine object. The values generated by std::random\_device are nondeterministic random numbers when possible, relying on random number generation devices, such as /dev/random. When such a device is not available, std::random\_device may employ a random number engine; however, the initial value generated should have sufficient randomness to serve as a seed value. |
| #include <random>  #include <iostream>    void f() {    std::random\_device dev;    std::mt19937 engine(dev());      for (**int** i = 0; i < 10; ++i) {      std::cout << engine() << ", ";    }  } |

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

|  |
| --- |
| **Principles(s):** [Name the principle and explain how it maps to this standard.] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Default-construction | Partially checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-MSC51 |  |
| Polyspace Bug Finder | R2020a | CERT C++:MSC51-CPP | Checks for:  Deterministic random output from constant seed  Predictable random output from predictable seed  Rule partially covered. |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-MSC51-a | Properly seed pseudorandom number generators |

### Coding Standard 10

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

| **Noncompliant Code** |
| --- |
| The function-try-block handler does not return a value, resulting in undefined behavior when an exception is thrown. |
| #include <vector>    std::**size\_t** f(std::vector<**int**> &v, std::**size\_t** s) try {    v.resize(s);    return s;  } catch (...) {  } |

| **Compliant Code** |
| --- |
| The exception handler of the function-try-block also returns a value. |
| #include <vector>    std::**size\_t** f(std::vector<**int**> &v, std::**size\_t** s) try {    v.resize(s);    return s;  } catch (...) {    return 0;  } |

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

|  |
| --- |
| **Principles(s):** Failing to return a value from a code path in a value-returning function results in undefined behavior that might be exploited to cause data 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 Suite | 6.9.0 | CertC++-MSC52 |  |
| Clang | 3.9 | -Wreturn-type | Does not catch all instances of this rule such as function-try-blocks |
| CodeSonar | 6.0p0 | LANG.STRUCT.MRS | Missing return statement |
| LDA tool suite | 9.7.1 | 2 S, 36 S | Fully implemented |

## 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 | Low | Unlikely | Low | 3 | 3 |
| STD-002-CPP | Medium | Unlikely | Medium | 4 | 3 |
| STD-003-CPP | High | Likely | Medium | 18 | 1 |
| STD-004-CPP | High | Probable | High | 6 | 2 |
| STD-005-CPP | High | Likely | Medium | 18 | 1 |
| STD-006-CPP | Low | Unlikely | Medium | 2 | 3 |
| STD-007-CPP | Low | Probable | Medium | 4 | 3 |
| STD-008-CPP | Medium | Unlikely | Medium | 4 | 3 |
| STD-009-CPP | Medium | Likely | Low | 18 | 1 |
| STD-010-CPP | Medium | Probable | Medium | 8 | 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 | When encrypted data is not being used. This should be applied because it will protect the data if there happens to be a breach while no-one is in the system. |
| Encryption at flight | When encrypted data is being transmitted. This makes sure that the data is safe while being transmitted between systems. |
| Encryption in use | When encrypted data is being used. This is set in place to ensure only the user is seeing the data that is being accessed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication ensures proper authorization to access a system is granted. This policy is applied so no un-authorized users are able to gain access. |
| Authorization | Authorization determines who gets what access to material. This policy applies so an employee only has access to what is needed to get their job done. |
| Accounting | Accounting measures the resources users use during their access to the system. This policy applies to make sure someone doesn’t access information they do not need to be in. |

**\***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 | 04/10/2021 | Update | Sean Churchill | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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