**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 – Definitions provided by https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | “Validate input from all untrusted data sources. Proper input validation can eliminate the vast majority of software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user controlled files.” |
| 1. Heed Compiler Warnings | “Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code [C MSC00-A, C++ MSC00-A]. Use static and dynamic analysis tools to detect and eliminate additional security flaws.” |
| 1. Architect and Design for Security Policies | “Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set.” |
| 1. Keep It Simple | “Keep the design as simple and small as possible [Saltzer 74, Saltzer 75]. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex.” |
| 1. Default Deny | “Base access decisions on permission rather than exclusion. This means that, by default, access is denied and the protection scheme identifies conditions under which access is permitted [Saltzer 74, Saltzer 75].” |
| 1. Adhere to the Principle of Least Privilege | “Every process should execute with the the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges [Saltzer 74, Saltzer 75].” |
| 1. Sanitize Data Sent to Other Systems | “Sanitize all data passed to complex subsystems [C STR02-A] such as command shells, relational databases, and commercial off-the-shelf (COTS) components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem.” |
| 1. Practice Defense in Depth | “Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment [Seacord 05].” |
| 1. Use Effective Quality Assurance Techniques | “Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions [Seacord 05].” |
| 1. Adopt a Secure Coding Standard | “Develop and/or apply a secure coding standard for your target development language and platform.” |

### 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.

***NOTE: I could not find anything within “Secure Coding in C and C++” or other given readings that denoted Ten Coding Standards. A lot of Name of standards here I put direct definitions for and most of the noncompliant and compliant code is copied from example code. I am unsure for most of these steps as I am a slow learner and I understand a little better with examples. I apologize for the inadequate quality of work or understanding, but I will try to do my best.***

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Proper ranging of integer data types must be adhered to so as to prevent wrapping (integer overflow) between signed and unsigned int conversions at their MAX and MIN |

| **Noncompliant Code** |
| --- |
| Truncation can occur when a value is too small to represent the result, and conversions can result in values out of range in the resulting type. |
| 1 unsigned long int ul = ULONG\_MAX;  2 signed char sc;  3 sc = (signed char)ul; /\* cast eliminates warning \*/ |

| **Compliant Code** |
| --- |
| Validate ranges when converting from an unsigned type to a signed type. |
| 1 unsigned long int ul = ULONG\_MAX;  2 signed char sc;  3 if (ul <= SCHAR\_MAX) {  4 sc = (signed char)ul; /\* use cast to eliminate warning \*/  5 }  6 else {  7 /\* handle error condition \*/  8 } |

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

| **Principles(s):** Validate Input Data – ensuring that the conversion has proper input |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Priority | High | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_C-INT31-l | Avoiding integer overflow |
| Cppcheck | 2.10 | memsetValueOutOfRange | The second argument to memset() |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | data types are declarations for variables. This determines the type and size of data associated with variables |

| **Noncompliant Code** |
| --- |
| This code fails to consider that the unsigned integer value will wrap around. (infinite loop) |
| 1 char a[MAX\_ARRAY\_SIZE] = /\* initialize \*/;  2 size\_t cnt = /\* initialize \*/;  3  4 for (unsigned int i = cnt-2; i >= 0; i--) {  5 a[i] += a[i+1];  6 } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| 1 char a[MAX\_ARRAY\_SIZE] = /\* initialize \*/;  2 size\_t cnt = /\* initialize \*/;  3  4 for (size\_t i = cnt-2; i != SIZE\_MAX; i--) {  5 a[i] += a[i+1];  6 } |

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

| **Principles(s):** Develop a secure Coding Standard – lets not allow division by zero errors |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_C-INT33-a | Avoid division by zero |
| Cppcheck | 2.10 | zerodiv  zerodivcond | Context for division of zero |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Use valid references, pointers, and iterators to reference elements of a basic\_string. Using an invalidated reference, pointer, or iterator will result in an undefined behavior. |

| **Noncompliant Code** |
| --- |
| C++: If a user inputs more than 11 characters, it will result in an out-of-bounds write |
| 1 #include <iostream>  2  3 int main(void) {  4 char buf[12];  5  6 std::cin >> buf;  7 std::cout << "echo: " << buf << '\n';  8 } |

| **Compliant Code** |
| --- |
| Eliminates the overflow in the previous example. |
| 1 #include <iostream>  2  3 int main(void) {  4 char buf[12];  5  6 std::cin.width(12);  7 std::cin >> buf; |

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

| **Principles(s):** Validate Input Data – for strings to get null terminated |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_C-STR31-e | Avoiding using unsafe string functions |
| Coverity | 2022.12 | STRING\_SIZE | string sizes |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Sanitizing data passed to complex subsystems and input validation ensures the mitigation of SQL Injection and other Injection attacks. |

| **Noncompliant Code** |
| --- |
| code passes raw input that may be potentially dangerous. |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| Parameters collection with Dynamic SQL |
| SqlDataAdapter myCommand = new SqlDataAdapter(  "SELECT au\_lname, au\_fname FROM Authors WHERE au\_id = @au\_id", conn);  SQLParameter parm = myCommand.SelectCommand.Parameters.Add("@au\_id",  SqlDbType.VarChar, 11);  Parm.Value = Login.Text; |

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

| **Principles(s):** Architect and Design for Security Policies - preventing an SQL Injection attack |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Priority | Medium | 5 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2022.1 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_ | implementation |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Allocate and free memory in the same module, at the same level of abstraction. Poor memory management can lead to security issues. |

| **Noncompliant Code** |
| --- |
| code shows that 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** |
| --- |
| wait until the memory is no longer required to deallocate |
| #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):** Heed Compiler Warnings – there are some instances where accessing freed memory will cause app crashing |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Priority | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_C-MEM31-a | Freed resources |
| Cppcheck | 2.10 | leakReturnValNotUsed | No memory allocation function values |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | assertions are used to test the value of a constant expression. Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in vulnerabilities. |

| **Noncompliant Code** |
| --- |
| any error code returned by this myGraphRoutine will be unhandled come final release. |
| myErr = myGraphRoutine(a, b);  /\* No Code to handle errors \*/  ASSERT(!myErr);  \_ASSERT(!myErr); |

| **Compliant Code** |
| --- |
| Because this approach evaluates assertions at compile time, there is no runtime penalty. |
| 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):** Developing a Coding Standard – limiting exposure |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | 1 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.9 | S3346 | Lack of side effects |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Use exceptions to check for errors that might occur such as input validation on parameters of public functions. |

| **Noncompliant Code** |
| --- |
| The function deallocates array and assigns the element counter which results in an undefined behavior. |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }      IntArray(const IntArray& that); // nontrivial copy constructor  IntArray& operator=(const IntArray &rhs) {  if (this != &rhs) {  delete[] array;  array = nullptr;  nElems = rhs.nElems;  if (nElems) {  array = new int[nElems];  std::memcpy(array, rhs.array, nElems \* sizeof(\*array));  }  }  return \*this;  }    // ...  }; |

| **Compliant Code** |
| --- |
| The function allocates new storage for the copy before changing the state of the object |
| #include <cstring>    class IntArray {  int \*array;  std::size\_t nElems;  public:  // ...    ~IntArray() {  delete[] array;  }    IntArray(const IntArray& that); // nontrivial copy constructor    IntArray& operator=(const IntArray &rhs) {  int \*tmp = nullptr;  if (rhs.nElems) {  tmp = new int[rhs.nElems];  std::memcpy(tmp, rhs.array, rhs.nElems \* sizeof(\*array));  }  delete[] array;  array = tmp;  nElems = rhs.nElems;  return \*this;  }    // ...  }; |

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

| **Principles(s):** Develop a Coding Standard – unhandled exceptions can cause the application to crash |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Priority | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions |
| RuleChecker | 23.04 | main-function-catch-all  early-catch-all | Partial checks |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Expressions** | [STD-008 -CPP] | Expressions are automatic variables that assume unexpected values if they are read before they are initialized, which can lead to undefined behavior. |

| **Noncompliant Code** |
| --- |
| variable is evaluated as part of an expression to print its value |
| #include <iostream>    void f() {  int i;  std::cout << i;  } |

| **Compliant Code** |
| --- |
| object is initialized prior to printing its value. |
| #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):** Use Effective Quality Assurance Techniques – using effective testing allows for easy fixing of issues |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Priority | High | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2023.1 | CERT\_CPP-ERR57-a | Freed resources (again) |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **One-definition rule** | [STD-009-CPP] | Don’t create more than one definition for a variable |

| **Noncompliant Code** |
| --- |
| Two different translation units define a class of the same name with different definitions |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| headers are used to introduce the object into both translation units |
| // S.h  struct S {  int a;  };    // a.cpp  #include "S.h"    // b.cpp  #include "S.h" |

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

| **Principles(s):** Keep It Simple – no reason to create more than one definition |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | High | High | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | type-compatibility  definition-duplicate  undefined-extern  undefined-extern-pure-virtual  external-file-spreading  type-file-spreading | Partially checked |
| Helix QAC | 2023.1 | C++1067, C++1509, C++1510 |  |
| RuleChecker | 22.10 | type-compatibility  definition-duplicate  undefined-extern  undefined-extern-pure-virtual  external-file-spreading  type-file-spreading | Partially checked |
| Polyspace Bug Finder | R2023a | CERT C++: DCL60-CPP | Partially checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Compile at highest warning level available** | [STD-010-CPP] | Compile code using the highest warning level possible for your compiler |

| **Noncompliant Code** |
| --- |
| Programmers commonly use the default warning specifier to restore previous warning messages after a message is temporarily disabled. |
| #pragma warning(disable:4705)  #pragma warning(disable:4706)  #pragma warning(disable:4707)  /\* Unnecessarily flagged code \*/  #pragma warning(default:4705)  #pragma warning(default:4706)  #pragma warning(default:4707) |

| **Compliant Code** |
| --- |
| The warnings should be saved and then restored after the unnecessarily flagged code |
| #pragma warning(push)  #pragma warning(disable:4705)  #pragma warning(disable:4706)  #pragma warning(disable:4707)  /\* Unnecessarily flagged code \*/  #pragma warning(pop) |

**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 | Medium | High Priority | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.25 |  |  |
| SonarQube C/C++ Plugin | 3.11 | S1762  S973 | Warns when the default warning specifier is used with #pragma warning.  Requires documentation of #pragma uses |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### 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.

Automation tools can be used to enforce and ensure compliance with policy standards by automatically monitoring and auditing systems for compliance violations. These tools can then scan systems for non-compliant configurations, detect unauthorized changes, and even generate alerts when violations are detected. Automation tools can also be used to enforce policy standards by automatically enforcing security controls such as access or encryption requirements. This can later on help organizations achieve and maintain compliance with policy standards more efficiently.

DevOps policies seem to excel in releasing new software that is up to par with current technological advances. With that being said, it is important to note that the main goal here is for the application of security principles in every phase of the Software Development Cycle, not just a singular one. This can be proven by example, such as understanding and documenting known attack patterns for the phase of assessing and planning. For the Verify and Test phase, I would like unit testing to be implemented. For the monitor phase, I think that some sort of security system be added so that unauthorized access to clearance on the system is rejected.

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | Low | Likely | Medium | Low | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Priority | Medium | 5 | 3 |
| STD-005-CPP | Medium | Priority | Medium | Medium | 2 |
| STD-006-CPP | Low | Unlikely | Low | 1 | 3 |
| STD-007-CPP | Low | Priority | Medium | Low | 3 |
| STD-008-CPP | Low | Priority | High | Low | 3 |
| STD-009-CPP | High | Unlikely | High | High | 3 |
| STD-0010-CPP | Medium | Likely | Medium | High Priority | 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 | A concept in where data that is not being actively utilized is encrypted. The goal of this policy is to protect data in the event of a breech and these files are stolen. |
| Encryption at flight | A concept where data as its moving around the network is being protected. The goal of this policy is to protect data while it’s arguably at its most vulnerable point and subject to the most exposure. |
| Encryption in use | A concept of protecting data when it’s actively being utilized. Protecting data-in-use is accomplished through the use of programming techniques like utilizing protected memory. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The process in which a server or application proves that you are who you say you are. This step is obviously important, as it determines who should or shouldn’t be granted access to the application. |
| Authorization | The process that occurs after Authentication and grants the level of access that you have to a system, its files, and its resources. New users should not inherently have access to any resources even if authenticated and only through extensive validation and role distribution will a user have access to specific data. |
| Accounting | The process of tracking changes to a particular system or resource. This is an important system to have as its both informative before and after a breach, as it can document the history of the incident before, current, and after. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [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 |