**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 | The process of validating input from various untrusted data sources is crucial to minimize or even eliminate potential software vulnerabilities. It is essential to maintain awareness while performing validation, especially when dealing with network interfaces, environmental variables, and external data sources. By ensuring thorough validation, we can significantly enhance the security of the software. |
| 1. Heed Compiler Warnings | To effectively address compiler warnings, it is advisable to compile code using the most stringent warning level settings. By doing so, any potential issues or inconsistencies in the code can be brought to light. It is essential to carefully review and modify the code whenever warnings are encountered during the compilation process. This proactive approach helps in preventing and resolving any underlying problems. |
| 1. Architect and Design for Security Policies | When developing your software, it is essential to prioritize and enhance security policies. This may involve implementing measures like dual access requirements, varying levels of security access depending on specific needs, and establishing predefined protocols to address any deviations from established policies. By carefully planning and designing your software with these considerations in mind, you can reinforce security and mitigate potential risks effectively. |
| 1. Keep It Simple | To ensure a smooth software lifecycle, it is advisable to maintain simplicity in both coding and design requirements. A complex design can lead to increased errors across various stages, including implementation and configuration. While security principles are crucial, incorporating them should be approached with a careful balance to avoid unnecessarily complicating the overall strategy. By keeping the coding and design straightforward, you can reduce the likelihood of errors and streamline the software development process effectively. |
| 1. Default Deny | By default, the access is set to deny, emphasizing a proactive security approach. The implementation of well-defined protocols becomes essential to determine the conditions under which access is granted or permissions are increased. These protocols act as guidelines that govern the decision-making process regarding user access and permissions within the software. Through this approach, potential security risks are mitigated, ensuring that access is only granted, or permissions are escalated under specific authorized circumstances. |
| 1. Adhere to the Principle of Least Privilege | In this context, the concept of "lowest level of access necessary" refers to granting users the minimum level of access required to fulfill their assigned tasks or mission. If there is a need for increased permission, it should be provided temporarily until the specific requirement is fulfilled, after which access is promptly reduced. This approach serves to limit the exposure of both external and internal threats, minimizing their opportunities to cause harm or compromise the system's security. By strictly controlling access privileges, potential risks can be effectively mitigated, safeguarding the integrity and confidentiality of the software. |
| 1. Sanitize Data Sent to Other Systems | To ensure data security, it is crucial to sanitize or cleanse the data in a way that limits its understanding to the calling process alone. This practice becomes especially important when dealing with complex subsystems like command shells and relational databases. By sanitizing the data, you can prevent hackers from exploiting vulnerabilities such as SQL injection attacks or unauthorized access to unused functionality. Implementing robust data sanitization techniques helps maintain data integrity and confidentiality, significantly reducing the risk of unauthorized access or malicious manipulation by external entities. |
| 1. Practice Defense in Depth | Developing multi-level overlapping defensive strategies is a key approach to enhancing security. By implementing this strategy, you create multiple layers of protection that work in tandem. In the event that one layer of security is compromised, the other overlapping layers are designed to kick in and prevent a breach or the exploitation of vulnerabilities. This approach significantly increases the overall resilience of your security measures, ensuring that even if one layer is breached, the remaining layers act as a robust defense system to mitigate the risk and maintain the integrity of your system. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance techniques play a crucial role in identifying and eliminating vulnerabilities within your software. Various methods such as penetration testing, source code audits, and thorough testing phases help uncover potential security weaknesses. Additionally, independent security reviews provide valuable insights from an external perspective, helping to identify issues that may have been overlooked internally. By employing these techniques and seeking external input, you can enhance the overall security posture of your software, proactively addressing vulnerabilities and ensuring a robust defense against potential threats. |
| 1. Adopt a Secure Coding Standard | Regardless of the programming language or platform you're working with, it is crucial to develop and implement a secure coding standard. This standard should encompass best practices, guidelines, and coding conventions that prioritize security throughout the development process. By creating a robust and well-thought-out secure coding standard, you can improve the efficiency and effectiveness of your software development efforts. Such a standard helps to minimize common security vulnerabilities, promotes consistent and secure coding practices among developers, and ultimately strengthens the overall security posture of your software. |

### 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):** Ensure enumeration values stay within specified limits to prevent buffer overflow vulnerabilities. Enforce data integrity through arithmetic validation, but be cautious of potential mathematical errors 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):** Pay attention to compiler warnings, design for security, keep code lightweight, employ effective quality assurance techniques, and follow a secure coding standard to prevent vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 | - |
| Astree | 20.10 | Reserved-identifier | Partially checked |
| 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, design with security policies, keep it simple, use effective quality assurance techniques, and adopt a secure coding standard. |
| --- |

**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' |
| PRQA QA-C++ | 4.4 | 0014 | - |
| 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 from 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 a previously owned pointer value in an unrelated smart pointer can lead to undefined behavior, potentially creating vulnerabilities in the system. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 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 obtained from the corresponding allocation function results in undefined behavior. This can potentially introduce exploitable vulnerabilities into the system. |
| --- |

**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 valuable diagnostic tool during the compilation process to identify software defects and mitigate vulnerabilities effectively. |
| --- |

**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 uncatchable exception can lead to program termination and potentially expose 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):** Include flush or positioning commands when alternating input and output operations to avoid undefined behavior. |
| --- |

**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** |
| --- | --- | --- |
| Value Inputs | [STD-009-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):** Architect and design with security policies, keep it simple, use effective quality assurance techniques, and adopt a secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2020a | CertC++: DCL58-CPP | Checks for modification of standard namespaces  Rule fully covered |
| Axivion Bauhaus Suite | 6.9.0 | CertC++-DCL58 | - |
| Parasoft C/C++ test | 2020.2 | CERT\_CPP-DCL58-a | Do not modify the standard namespaces 'std' and 'posix |
| PRQA QA-C++ | 4.4 | 4032, 4035, 4631 | - |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Namespace modifications | [STD-010-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):** Failing to return a value in a value-returning function leads to undefined behavior, potentially causing 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

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.

The automation framework implemented in the DevOps process is robust and well-integrated across all its stages. Similar to other offerings, this process further extends its automation capabilities to encompass post-release updates and modifications, enabling seamless deployment of new versions in the future.

### 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 | Data at rest encryption entails the secure encoding of data as it is stored, and its subsequent decryption when retrieved for use. The utilization of an encryption key during the storage process safeguards the data against unauthorized access. This protective measure should be employed for all types of data, regardless of its sensitivity level, as unauthorized access by malicious individuals can result in significant harm. |
| Encryption at flight | Data in transit encryption involves the secure encoding of data during its transmission. The specific method of data transfer determines the application of this encryption. When using web browsers, it is important to implement secure protocols to ensure the encryption of data. For sending emails, encrypting the content before transmission and utilizing digital signatures can enhance security. These measures help protect the confidentiality and integrity of the data while it is in transit. |
| Encryption in use | Data in-use encryption focuses on safeguarding data while it is being actively utilized in computer memory. One approach to achieve this is by employing password-protected profiles that secure the memory associated with each individual profile. By implementing this measure, the data remains protected even during its active processing, mitigating the risk of unauthorized access or tampering. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | To establish user identity, a multi-step authentication process is employed, incorporating various security measures. These measures include the use of user IDs, passwords, and additional layers of security such as secure tokens, Common Access Cards (CAC), PINs, and other forms of dual authentication. This comprehensive approach ensures a robust verification process, reducing the risk of unauthorized access and enhancing overall security. |
| Authorization | After successful authentication and gaining access, users are granted specific and limited permissions within the system. System administrators assign authorized access to particular drives, folders, programs, and other data based on user roles and responsibilities. This controlled approach ensures that users can only access the resources necessary for their designated tasks, enhancing security by minimizing the risk of unauthorized data exposure or modification. System administrators play a crucial role in maintaining these access controls and regularly reviewing and updating them as needed. |
| Accounting | After authentication and authorization, continuous monitoring and recording of user activities are necessary to ensure system integrity and security. |

**\***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 | 06/11/2023 | Project One | Luke Peters | Luke Peters |
| 1.1 | 06/11/2023 | Final Revision | Luke Peters | Luke Peters |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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