**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 | In order to prevent software vulnerabilities, input data has to be validated. Malicious sources could access private data if input data is not accounted for properly. External sources should also be accounted for such as network interfaces and environmental variables. |
| 1. Heed Compiler Warnings | Compiler warnings are important. If compiling issues are left unchecked it could create security breaches. Utilizing analysis tools (static and dynamic) can help detect potential bugs and correct them. Being mindful of compiler warnings can improve security and prevent an attack on the system. |
| 1. Architect and Design for Security Policies | Software architect and design should be created in order to enforce Security Policies. This helps to improve software quality and prevent security risks. |
| 1. Keep It Simple | Simplicity is preferred over complexity. Complex designs lead to more errors in configuration and implementation. Compilation errors could also occur and take longer to resolve if the code is complex. Complex designs are also harder to secure from a security stand point. |
| 1. Default Deny | Access decisions are based more on permissions then exclusion. By default, access will be denied and the permissions in place will determine if access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Processes should process with the minimum or least set of privileges. If a permission is elevated it should only be accessed for a limited time. Doing so, limits the possibility that an attack will be able to take advantage of the elevated privileges and execute arbitrary code. |
| 1. Sanitize Data Sent to Other Systems | Command shells and relational database data should be sanitized before being passed through more complex subsystems. This cannot be caught within validating the input because the complex subsystem does not know the context of the call being made. Thus, in order to prevent attacks from conducting injection attacks, it is important to sanitize data sent to other systems. |
| 1. Practice Defense in Depth | Multiple defense strategies help to create a layered security system. If one layer of security were to be compromised then the second layer of security should catch this security flaw. If it does not prevent it, it will limit the consequences of the successful exploit. |
| 1. Use Effective Quality Assurance Techniques | Quality Assurance helps to assure that the techniques used are effective at eliminating vulnerabilities or identifying them. Source code audits and penetration testing should be used to test the quality of the design. |
| 1. Adopt a Secure Coding Standard | Adopt a secure coding standard sets a secure coding standard. Especially in terms of development language and platform. This will help eliminate and prevent errors that could impact software security. |

### 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] | Defines variadic function that could lead to security risks, because an argument goes unchecked and is passed. |

| **Noncompliant Code** |
| --- |
| The function below reads values until 0 is found. If not, after two arguments there is an increased risk of vulnerability seeing as it will result in undefined behavior. |
| #include <cstdarg>    int add(int first, int second, ...) {  int r = first + second;  va\_list va;  va\_start(va, second);  while (int v = va\_arg(va, int)) {  r += v;  }  va\_end(va);  return r;  } |

| **Compliant Code** |
| --- |
| Recursive expansion is not used in this set of code. Instead, it expands the function parameter pack values as part of the braced initializer list. Narrowed conversion cannot be used in a braced initializer list. This design does not result in undefined behavior. |
| #include <type\_traits>    **template** <**typename** Arg, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Arg s) { **return** f + s; }    **template** <**typename** Arg, **typename**... Ts, **typename** std::enable\_if<std::is\_integral<Arg>::value>::type \* = nullptr>  **int** add(Arg f, Ts... rest) {  **return** f + add(rest...);  } |

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

| **Principles(s):**  Quality Assurance: ensures that the code has a solid foundation and does not create undefined behaviors. This can be done by running a variety of test scenarios and values through the code. In this case, this is important to ensure that the code is reading values until 0 is found to prevent undefined behavior.  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **function-ellipsis** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-dcl50-cpp | Checked by clang-tidy. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.2p0 | **LANG.STRUCT.ELLIPSIS** | Ellipsis |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Incorrectly defined identifiers(reserved) create issues seeing as it will not be stored properly. |

| **Noncompliant Code** |
| --- |
| Conditional guards against inclusions of a header file. Reserve names are used for header guards although this can create issues and may even clash with standard C++ library information. In this case, naming standards were not met and undefined behavior occurs. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| Leading and trailing underscores are removed from the code. Naming Standards are met and undefined behavior no longer occurs. |
| #ifndef MY\_HEADER\_H  #define MY\_HEADER\_H    // Contents of <my\_header.h>    #endif // MY\_HEADER\_H |

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

| **Principles(s):**  Heed Compiler Warnings: Compiler warnings are important. If compiling issues are left unchecked it could create security breaches.  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error. Naming standards should be maintained to avoid errors in the code, leading and trailing underscores have the possibility to create these issues.  Keep it simple: Simple designs lead to less errors. Reduces the amount of time spend on debugging and is easy to manage. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **reserved-identifier** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL51** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wreserved-id-macro -Wuser-defined-literals | The -Wreserved-id-macro flag is not enabled by default or with -Wall, but is enabled with -Weverything. This flag does not catch all instances of this rule, such as redefining reserved names. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.2p0 | **LANG.ID.NU.MK**  **LANG.STRUCT.DECL.RESERVED** | Macro name is C keyword  Declaration of reserved name |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | A reference type should not be qualified with const or volatile. Compiler is expected to produce a diagnostic. If not, then it could lead to undefined behaviors. Especially if cv-qualifying is being utilized. |

| **Noncompliant Code** |
| --- |
| Const is being used to reference char instead of a const-qualified char. The error here is that it cannot be applied as a reference as it is written because it will cause undefined behaviors. |
| #include <iostream>    void f(char c) {  char &const p = c;  p = 'p';  std::cout << c << std::endl;  } |

| **Compliant Code** |
| --- |
| Const qualifier is removed from the code. In return, errors will not occur. |
| #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):**  Heed Compiler Warnings: Compiler warnings are important. If compiling issues are left unchecked it could create security breaches. In this case, a reference was made that causes undefined behaviors so it is important to follow compiler warnings, otherwise undefined behaviors would have occurred.  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error. Strings should follow a standard so that when references are made the code does compile incorrectly.  Keep it simple: Simple designs lead to less errors. Reduces the amount of time spend on debugging and is easy to manage. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL52-a** | Never qualify a reference type with 'const' or 'volatile' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: DCL52-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl52cpp.html) | Checks for:   * const-qualified reference types * Modification of const-qualified reference types   Rule fully covered. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **0014** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 |  | Clang checks for violations of this rule and produces an error without the need to specify any special flags or options. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CCP] | Code should be understood in one standard way, not in syntactically ambiguous declarations. |

| **Noncompliant Code** |
| --- |
| The argument below can either declare an object, calling a single argument converting constructor. But it also can be translated as a declaring object m and constructing it. |
| #include <mutex>    static std::mutex m;  static int shared\_resource;    void increment\_by\_42() {  std::unique\_lock<std::mutex>(m);  shared\_resource += 42;  } |

| **Compliant Code** |
| --- |
| Converting Constructor is called properly and the lock has an identifier. |
| #include <mutex>    static std::mutex m;  static int shared\_resource;    void increment\_by\_42() {  std::unique\_lock<std::mutex> lock(m);  shared\_resource += 42;  } |

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

| **Principles(s):**  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error. The standard in the non-compliant code was not secure seeing as it can be interpreted in two different ways.  Keep it simple: Simple designs lead to less errors. Reduces the amount of time spend on debugging and is easy to manage.  Validating Input Data: Ensures that the proper inputs are entered and flags inputs outside its range.  Use Effective Quality Assurance Techniques: Quality Assurance helps to assure that the techniques used are effective at eliminating vulnerabilities or identifying them. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **296 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL53-a** **CERT\_CPP-DCL53-b** **CERT\_CPP-DCL53-c** | Parameter names in function declarations should not be enclosed in parentheses Local variable names in variable declarations should not be enclosed in parentheses Avoid function declarations that are syntactically ambiguous |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: DCL53-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl53cpp.html) | Checks for declarations that can be confused between:   * Function and object declaration * Unnamed object or function parameter declaration   Rule fully covered. |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2502, 2510** |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CCP] | Overload deallocation and allocation functions are utilized as a pair within a scope. If not utilized like a pair, it will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Deallocation function is not declared at global scale, but allocation is overloaded at global scale which will result in an error. |
| #include <Windows.h>  #include <new>    void \*operator new(std::size\_t size) noexcept(false) {  static HANDLE h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  if (h) {  return ::HeapAlloc(h, 0, size);  }  throw std::bad\_alloc();  }    // No corresponding global delete operator defined. |

| **Compliant Code** |
| --- |
| Overload deallocation and allocation functions are declared as a pair, therefore the overload condition will not occur. |
| #include <Windows.h>  #include <new>    class HeapAllocator {  static HANDLE h;  static bool init;    public:  static void \*alloc(std::size\_t size) noexcept(false) {  if (!init) {  h = ::HeapCreate(0, 0, 0); // Private, expandable heap.  init = true;  }    if (h) {  return ::HeapAlloc(h, 0, size);  }  throw std::bad\_alloc();  }    static void dealloc(void \*ptr) noexcept {  if (h) {  (void)::HeapFree(h, 0, ptr);  }  }  };    HANDLE HeapAllocator::h = nullptr;  bool HeapAllocator::init = false;    void \*operator new(std::size\_t size) noexcept(false) {  return HeapAllocator::alloc(size);  }    void operator delete(void \*ptr) noexcept {  return HeapAllocator::dealloc(ptr);  } |

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

| **Principles(s):**  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error. Understanding pairs within a scope will assist maintaining a secure coding standard.  Keep it simple: Simple designs lead to less errors. Reduces the amount of time spend on debugging and is easy to manage.  Validating Input Data: Ensures that the proper inputs are entered and flags inputs outside its range.  Use Effective Quality Assurance Techniques: Quality Assurance helps to assure that the techniques used are effective at eliminating vulnerabilities or identifying them. |
| --- |

**Threat Level**

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

Automation

| Tool | Version | Checker | Description Tool |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL54-a** | Always provide new and delete together |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: DCL54-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl54cpp.html) | Checks for mismatch between overloaded operator new and operator delete (rule fully covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **2160** |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **new-delete-pairwise** | Partially checked |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CCP] | Data passing will need to be verified in order to prevent data leaks when passing the class object to a trusted boundary. |

| **Noncompliant Code** |
| --- |
| Data transferred may contain sensitive data in this design and is not serialized. |
| struct test {  int a;  char b;  int c;  };    // Safely copy bytes to user space  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {  test arg{1, 2, 3};  copy\_to\_user(usr\_buf, &arg, sizeof(arg));  } |

| **Compliant Code** |
| --- |
| Serialized structure is applied to the design before data is copied to prevent data leaks. |
| #include <cstddef>  #include <cstring>    struct test {  int a;  char b;  int c;  };    // Safely copy bytes to user space.  extern int copy\_to\_user(void \*dest, void \*src, std::size\_t size);    void do\_stuff(void \*usr\_buf) {  test arg{1, 2, 3};  // May be larger than strictly needed.  unsigned char buf[sizeof(arg)];  std::size\_t offset = 0;    std::memcpy(buf + offset, &arg.a, sizeof(arg.a));  offset += sizeof(arg.a);  std::memcpy(buf + offset, &arg.b, sizeof(arg.b));  offset += sizeof(arg.b);  std::memcpy(buf + offset, &arg.c, sizeof(arg.c));  offset += sizeof(arg.c);    copy\_to\_user(usr\_buf, buf, offset /\* size of info copied \*/);  } |

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

| **Principles(s):**  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error.  Keep it simple: Simple designs lead to less errors. Reduces the amount of time spend on debugging and is easy to manage.  Validating Input Data: Ensures that the proper inputs are entered and flags inputs outside its range.  Use Effective Quality Assurance Techniques: Quality Assurance helps to assure that the techniques used are effective at eliminating vulnerabilities or identifying them.  Practice Defense in Depth: the code above is serialized, this a layer of security to prevent data leaks. If one layer of security were to be compromised then the second layer of security should catch this security flaw |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL55** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++4941, C++4942, C++4943** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL55-a** | A pointer to a structure should not be passed to a function that can copy data to the user space |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CCP] | During Initialization avoid cycles of static objects. Undefined behavior will occur if the function is reentered during initialization. |

| **Noncompliant Code** |
| --- |
| Implements factorial function while using caching. Undefined behavior occurs due to the initialization of the static array since it uses recursion. |
| #include <stdexcept>    int fact(int i) noexcept(false) {  if (i < 0) {  // Negative factorials are undefined.  throw std::domain\_error("i must be >= 0");  }    static const int cache[] = {  fact(0), fact(1), fact(2), fact(3), fact(4), fact(5),  fact(6), fact(7), fact(8), fact(9), fact(10), fact(11),  fact(12), fact(13), fact(14), fact(15), fact(16)  };    if (i < (sizeof(cache) / sizeof(int))) {  return cache[i];  }    return i > 0 ? i \* fact(i - 1) : 1;  } |

| **Compliant Code** |
| --- |
| Removes static cache array, eliminating the undefined behavior that was occurring. |
| #include <stdexcept>    int fact(int i) noexcept(false) {  if (i < 0) {  // Negative factorials are undefined.  throw std::domain\_error("i must be >= 0");  }    // Use the lazy-initialized cache.  static int cache[17];  if (i < (sizeof(cache) / sizeof(int))) {  if (0 == cache[i]) {  cache[i] = i > 0 ? i \* fact(i - 1) : 1;  }  return cache[i];  }    return i > 0 ? i \* fact(i - 1) : 1;  } |

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

| **Principles(s):**  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error.  Keep it simple: Simple designs lead to less errors. Reduces the amount of time spend on debugging and is easy to manage.  Validating Input Data: Ensures that the proper inputs are entered and flags inputs outside its range.  Use Effective Quality Assurance Techniques: Quality Assurance helps to assure that the techniques used are effective at eliminating vulnerabilities or identifying them. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.2p0 | **LANG.STRUCT.INIT.CYCLE**  **LANG.STRUCT.INIT.UNORDERED** | Initialization Cycle  Unordered Initialization |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2021.2 | **C++1552, C++1554, C++1704** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **6 D** | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL56-a** | Avoid initialization order problems across translation units by replacing non-local static objects with local static objects |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-008-CCP] | Standard namespaces should not be modified. If a new declaration is misused it could result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Declaring x in the namespace std results in undefined behavior. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| Declaration is placed in a namespace without a reserved name eliminating the undefined behavior. |
| namespace nonstd {  int x;  } |

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

| **Principles(s):**  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error.  Keep it simple: Simple designs lead to less errors. Reduces the amount of time spend on debugging and is easy to manage.  Use Effective Quality Assurance Techniques: Quality Assurance helps to assure that the techniques used are effective at eliminating vulnerabilities or identifying them. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2021.4 | [CERT.DCL.STD\_NS\_MODIFIED](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: DCL58-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl58cpp.html) | Checks for modification of standard namespaces (rule fully covered) |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | **4032, 4035, 4631** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CCP] | Unnamed namespace cannot be defined. This will result in an undefined behavior. |

| **Noncompliant Code** |
| --- |
| Variable v is defined within an unnamed namespace. This will cause each translation unit to operate on its own instance of the variable. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    namespace {  int v;  }    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    void f() {  std::cout << "f(): " << v << std::endl;  v = 42;  // ...  }    // b.cpp  #include "a.h"  #include <iostream>    void g() {  std::cout << "g(): " << v << std::endl;  v = 100;  }    int main() {  extern void f();  f(); // Prints v, sets it to 42  g(); // Prints v, sets it to 100  f();  g();  } |

| **Compliant Code** |
| --- |
| Variable V is defined in one translation unit, but is visible to each translation unit externally. |
| // a.h  #ifndef A\_HEADER\_FILE  #define A\_HEADER\_FILE    extern int v;    #endif // A\_HEADER\_FILE    // a.cpp  #include "a.h"  #include <iostream>    int v; // Definition of global variable v    void f() {  std::cout << "f(): " << v << std::endl;  v = 42;  // ...  }    // b.cpp  #include "a.h"  #include <iostream>    void g() {  std::cout << "g(): " << v << std::endl;  v = 100;  }    int main() {  extern void f();  f(); // Prints v, sets it to 42  g(); // Prints v, sets it to 100  f(); // Prints v, sets it back to 42  g(); // Prints v, sets it back to 100  } |

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

| **Principles(s):**  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error.  Keep it simple: Simple designs lead to less errors. Reduces the amount of time spend on debugging and is easy to manage.  Use Effective Quality Assurance Techniques: Quality Assurance helps to assure that the techniques used are effective at eliminating vulnerabilities or identifying them. In this case, Quality Assurance would catch that each translation of v was calling its own instance. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **286 S, 512 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL59-a** | There shall be no unnamed namespaces in header files |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: DCL59-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl59cpp.html) | Checks for unnamed namespaces in header files (rule fully covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **unnamed-namespace-header** | Fully checked |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CCP] | Do not violate the one-definition rule. Meaning the program can contain exactly one definition of every non inline function or variable. If OCR is violated it will result in undefined behavior. |

| **Noncompliant Code** |
| --- |
| Two translation units are defining a class with different definitions. Since they are defined with different tokens it will result in undefined behavior. |
| // a.cpp  struct S {  int a;  };    // b.cpp  class S {  public:  int a;  }; |

| **Compliant Code** |
| --- |
| Header file introduces object into both translation units. Fixing the undefined behavior. |
| // 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):**  Adopt a secure coding standard: Maintaining a secure coding procedure will limit errors that could impact software security. Reduces the amount of time spend debugging an error.  Keep it simple: Simple designs lead to less errors. Reduces the amount of time spend on debugging and is easy to manage.  Use Effective Quality Assurance Techniques: Quality Assurance helps to assure that the techniques used are effective at eliminating vulnerabilities or identifying them. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **286 S, 287 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL60-a** | A class, union or enum name (including qualification, if any) shall be a unique identifier |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2021b | [CERT C++: DCL60-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl60cpp.html) | Checks for inline constraints not respected (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | **type-compatibility** **definition-duplicate** **undefined-extern** **undefined-extern-pure-virtual** **external-file-spreading** **type-file-spreading** | Partially checked |

### Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

### Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

### Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

### Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

### Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

Automation is essential throughout the process and after the final release of the product. In order to improve the product, it must always be updated and changed as new data is discovered. In return, this advances product quality and efficiency.

### 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 | Probable | Medium | **P12** | **L1** |
| STD-002-CPP | Low | Unlikely | Low | **P3** | **L3** |
| STD-003-CPP | Low | Unlikely | Low | **P3** | **L3** |
| STD-004-CPP | Low | Unlikely | Medium | **P2** | **L3** |
| STD-005-CPP | Low | Probable | Low | **P6** | **L2** |
| STD-006-CPP | Low | Unlikely | High | **P1** | **L3** |
| STD-007-CPP | Low | Unlikely | Medium | **P2** | **L3** |
| STD-008-CPP | High | Unlikely | Medium | **P6** | **L2** |
| STD-009-CPP | Medium | Unlikely | Medium | **P4** | **L3** |
| STD-010-CPP | High | Unlikely | High | **P3** | **L3** |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest prevents the attacker from obtaining access to unencrypted data by encrypting the data on disk. Symmetric encryption keys help maintain security in storage. If the user does not have the encryption keys even if the storage is stolen, they may not be able to access the information inside the device. |
| Encryption at flight | Encryption in flight is the process in which data is encrypted while the data is being transmitted. This type of encryption applies in situations such as sending emails. Ensuring the data within web browsers is secure is also essential to ensure that the information being sent is secure. In both these cases, the data is being encrypted as it is being transmitted. |
| Encryption in use | Encryption in use is the process in which sensitive data is not left unsecured. Since the data is stored in memory. Unlike encryption in rest or at flight, this policy ensures that data is secure no matter the cycle or location. This can be maintained by utilizing passwords or authentication. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process in which proves the identity of the user. This is done so by verifying the password, ID, or other security methods such as security tokens. The policy helps protect the system from suspicious users that may be trying to access the database. |
| Authorization | Once the user is verified, the system will determine what the user is authorized to do within the system. The user may only be able to access a specific set of drivers and files. If they need access to files that their permissions may not allow for, they may have to contact system admins for assistance. |
| Accounting | Accounting is in charge of monitoring user usage. This may monitor the amount of data being sent or received while the user is on the network. The policy assists in catching suspicious behaviors or catch where a user may have encountered an issue. |

**\***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
  + - Keep It Simple: 4 – Operating systems logs should be simple, but secure to access. Therefore, a simple solution that is easy to understand and protect will work better than an overly complicated solution.
    - Default Deny: 5 – Protects the operating system logs from unauthorized users. Denies access if the user is unauthorized. Authorized users will only be able to access operating system logs.
    - Adhere to the Principle of Least Privilege: 6 – If the user needs to access to logs, but does not have permissions. Then the user must speak to system admins to access the necessary files. This prevents unauthorized users from accessing information much like default deny.
    - Practice Defense In Depth: 8 - Multiple layers of protection could help safeguard the system especially when it comes to the operating system and its logs.
    - Adopt a Secure Coding Standard: 10 - The design of the code is critical in order to maintain quality security and trouble shoot or detect suspicious activity.
* Firewall logs
  + - Keep It Simple: 4 - Firewall logs should be simple, but secure to access. Therefore, a simple solution that is easy to understand and protect will work better than an overly complicated solution.
    - Default Deny: 5 - Protects the firewall logs from unauthorized users. Denies access if the user is unauthorized. Authorized users will only be able to access firewall logs or change firewall settings.
    - Adhere to the Principle of Least Privilege: 6 – If the user needs to access to logs, but does not have permissions. Then the user must speak to system admins to access the necessary files. This prevents unauthorized users from accessing information much like default deny.
    - Sanitize Data Sent to Other System: 7 – Ensures that systems are sanitized before sending code. If code isn’t checked then this can create vulnerabilities.
    - Practice Defense In Depth: 8 – Multiple layers of protection could help safeguard the system especially when it comes to fire wall protection and its logs.
    - Adopt a Secure Coding Standard: 10 – The design of the code is critical in order to maintain quality security and trouble shoot or detect suspicious activity.
* Anti-malware logs
  + - Keep It Simple: 4 - Anti-malware logs should be simple, but secure to access. Therefore, a simple solution that is easy to understand and protect will work better than an overly complicated solution.
    - Default Deny: 5 - Protects the firewall logs from unauthorized users. Denies access if the user is unauthorized. Authorized users will only be able to access firewall logs or change firewall settings.
    - Adhere to the Principle of Least Privilege: 6 -– If the user needs to access to logs, but does not have permissions. Then the user must speak to system admins to access the necessary files. This prevents unauthorized users from accessing information much like default deny.
    - Sanitize Data Sent to Other System: 7 - Ensures that systems are sanitized before sending code. If code isn’t checked then this can create vulnerabilities.
    - Practice Defense In Depth: 8 - Multiple layers of protection could help safeguard the system especially when it comes to anti-malware protection and its logs.
    - Adopt a Secure Coding Standard: 10 - The design of the code is critical in order to maintain quality security and trouble shoot or detect suspicious activity.

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 | 2/13/2022 | Project One | Cheyenne Heckman | Cheyenne Heckman |
| [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 |