**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 | Process in which all data is treated as untrusted until properly validated and tested. This practice will reduce vulnerabilities. |
| 1. Heed Compiler Warnings | This principle requires us to compile code at the highest warning level for the compiler used and eliminate warnings by modifying the code. |
| 1. Architect and Design for Security Policies | This principle requires that there is a set template for how you will manage the security of the software. By planning and implementing polices that are reviewed and enforced you mitigate most issues before they occur. |
| 1. Keep It Simple | The principle here is to keep your design as simple as possible. Implementing complex designs increases the likelihood of errors which compromise security. |
| 1. Default Deny | This best practice assumes a deny as the default. That way permissions have to be explicitly set. |
| 1. Adhere to the Principle of Least Privilege | Least privilege gives the lowest permissions needed to complete the task. The least privilege forces any elevated permissions be explicitly set or approved and also decreases the damage an elevated permission could inflict due to error or unauthorized use. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. |
| 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. |
| 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. |
| 1. Adopt a Secure Coding Standard | Develop and/or apply a secure coding standard for your target development language and platform. |

## https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices

## 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** |
| --- | --- | --- |
| **Declarations and Initialization** | [STD-001-DCL] | The standard ensures that all declarations and initializations follow best practices and remove potential exploits’ |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses a C-style variadic function to add a series of integers together. The function reads arguments until the value 0 is found. Calling this function without passing the value 0 as an argument (after the first two arguments) results in undefined behavior. Furthermore, passing any type other than an int also results 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** |
| --- |
| a variadic function using a function parameter pack is used to implement the add() function, allowing identical behavior for call sites. |
| #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):** Automation |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description** |
| --- | --- | --- | --- |
| [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) | **6.9.0** | **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) | **6.0p0** | **LANG.STRUCT.ELLIPSIS** | **Ellipsis** |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | **9.7.1** | **41 S** | **Fully Implemented** |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Expressions** | [STD-002-EXP] | Focus on reviewing and checking the expression to ensure that best practices are used |

| **Noncompliant Code** |
| --- |
| Derived objects is created and the pointer is stored in a Base \*. Despite Base::~Base() being declared virtual, it still results in undefined behavior. Further, attempting to perform pointer arithmetic on the static type Base \* violates CTR56-CPP. Do not use pointer arithmetic on polymorphic objects. |
| [struct Base {    virtual ~Base() = default;  };    struct Derived final : Base {};    void f() {     Base \*b = new Derived[10];     // ...     delete [] b;  } |

| **Compliant Code** |
| --- |
| The static type of b is Derived \*, which removes the undefined behavior when indexing into the array as well as when deleting the pointer. |
| struct Base {    virtual ~Base() = default;  };    struct Derived final : Base {};    void f() {     Derived \*b = new Derived[10];     // ...     delete [] b;  } |

**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** |
| --- | --- | --- | --- | --- |
| EXP50-CPP | EXP50-CPP | EXP50-CPP | EXP50-CPP | EXP50-CPP |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC++-EXP50** |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Rose) |  |  | Can detect simple violations of this rule. It needs to examine each expression and make sure that no variable is modified twice in the expression. It also must check that no variable is modified once, then read elsewhere, with the single exception that a variable may appear on both the left and right of an assignment operator |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | v7.5.0 | **EVALUATION\_ORDER** | Can detect the specific instance where a statement contains multiple side effects on the same value with an undefined evaluation order because, with different compiler flags or different compilers or platforms, the statement may behave differently |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Integers** | [STD-003-INT] | Ensure that any integers used to not cause errors or introduce exploits. This requires reviewing integers and checking for implementations that are either not allowed or do not mathematically compute. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example can result in a signed integer overflow during the addition of the signed operands si\_a and si\_b: |
| void func(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum = si\_a + si\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This compliant solution ensures that the addition operation cannot overflow, regardless of representation: |
| #include <limits.h>    void f(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum;    if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||        ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {      /\* Handle error \*/    } else {      sum = si\_a + si\_b;    }    /\* ... \*/  } |

**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** |
| --- | --- | --- | --- | --- |
| INT50-CPP | Medium | Unlikely | Medium | **P4** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC++-INT50** | [Insert text.] |
| [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 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Containers** | [STD-004-CTR] | Review containers for possible errors and eploits ensures container rules are followed and avoid problematic implementations. |

| **Noncompliant Code** |
| --- |
| f() accepts an array of S objects as its first parameter. However, main() passes an array of T objects as the first argument to f(), which results in undefined behavior due to the pointer arithmetic used within the for loop. |
| #include <iostream>    // ... definitions for S, T, globI, globD ...    void f(const S \*someSes, std::size\_t count) {  for (const S \*end = someSes + count; someSes != end; ++someSes) {  std::cout << someSes->i << std::endl;  }  }    int main() {  T test[5];  f(test, 5);  } |

| **Compliant Code** |
| --- |
| Instead of having an array of objects, an array of pointers solves the problem of the objects being of different sizes, as in this compliant solution. |
| #include <iostream>   void f(const S \* const \*someSes, std::**size\_t** count) {    for (const S \* const \*end = someSes + count; someSes != end; ++someSes) {      std::cout << (\*someSes)->i << std::endl;    }  }    **int** main() {    S \*test[] = {new T, new T, new T, new T, new T};    f(test, 5);    for (auto v : test) {      delete v;    }  } |

**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** |
| --- | --- | --- | --- | --- |
| CTR50-CPP | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **LANG.MEM.BO LANG.MEM.BU LANG.MEM.TO LANG.MEM.TU LANG.MEM.TBA LANG.STRUCT.PBB LANG.STRUCT.PPE** | Buffer overrun Buffer underrun Type overrun Type underrun Tainted buffer access Pointer before beginning of object Pointer past end of object |
| [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-MEM] | Review and implement best practices to prevent improper implementation of memory that would put the code at risk for exploits |

| **Noncompliant Code** |
| --- |
| an array of int is created using ::operator new[](std::size\_t) and the results of the allocation are not checked. The function is marked as no except, so the caller assumes this function does not throw any exceptions. Because ::operator new[](std::size\_t) can throw an exception if the allocation fails, it could lead to abnormal termination of the program |
| #include <cstring>    void f(const **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = new **int**[size];    std::**memcpy**(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

| **Compliant Code** |
| --- |
| using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles the error condition appropriately when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    void f(const **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = new (std::nothrow) **int**[size];    if (!copy) {      // Handle error      return;    }    std::**memcpy**(copy, array, size \* sizeof(\*copy));    // ...    delete [] copy;  } |

**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** |
| --- | --- | --- | --- | --- |
| MEM50-CPP | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **dangling\_pointer\_use** | [Insert text.] |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC++-MEM50** | [Insert text.] |
| [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** |
| --- | --- | --- |
| **Characters and strings** | [STD-006-STR] | The standard forces the review of characters and strings follow best practices and logical implementation to avoid exploits and errors that can compromise the code. |

| **Noncompliant Code** |
| --- |
| the char pointer str is initialized to the address of a string literal. Attempting to modify the string literal is undefined behavior |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| This code creates a copy of the string literal in the space allocated to the character array str. The string stored in str can be modified safely. |
| char str[] = "string literal";  str[0] = 'S'; |

**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** |
| --- | --- | --- | --- | --- |
| STR50-CPP | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | [Insert text.] |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO LANG.MEM.TO** | [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-ERR] | Exceptions and errors should be well though out and implemented through in order to catch problems before they exist and have the fore thought to plan on potential exploits. These exceptions should handle all security risks that could arise with the given code. |

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

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

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

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

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| ERR50-CPP | Probable | Medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | [Insert text.] |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **stdlib-use** | [Insert text.] |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | **BADFUNC.ABORT BADFUNC.EXIT** | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | [STD-008-OOP] | This approach separates the code into smaller pieces. This decreases the codes complexity allows smaller parts to be better reviewed and tested for errors. The organization also allows for code to be found and revisited. |

| **Noncompliant Code** |
| --- |
| an object of the derived Manager type is passed by value to a function accepting a base Employee type. Consequently, the Manager objects are sliced, resulting in information loss and unexpected behavior when the print() function is called. |
| [#include <iostream>  #include <string>    class Employee {    std::string name;    protected:    virtual void print(std::ostream &os) const {      os << "Employee: " << get\_name() << std::endl;    }    public:    Employee(const std::string &name) : name(name) {}    const std::string &get\_name() const { return name; }    friend std::ostream &operator<<(std::ostream &os, const Employee &e) {      e.print(os);      return os;    }  };    class Manager : public Employee {    Employee assistant;    protected:    void print(std::ostream &os) const override {      os << "Manager: " << get\_name() << std::endl;      os << "Assistant: " << std::endl << "\t" << get\_assistant() << std::endl;    }    public:    Manager(const std::string &name, const Employee &assistant) : Employee(name), assistant(assistant) {}    const Employee &get\_assistant() const { return assistant; }  };    void f(Employee e) {    std::cout << e;  }    **int** main() {    Employee coder("Joe Smith");    Employee typist("Bill Jones");    Manager designer("Jane Doe", typist);      f(coder);    f(typist);    f(designer);  } |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

**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** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | **virtual-call-in-constructor invalid\_function\_pointer** | [Insert text.] |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 6.9.0 | **CertC++-OOP50** | [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** |
| --- | --- | --- |
| Concurrency | [STD-009-CON] | You want to limit concurrency to not allow exploits that may allow permission or access to unauthorized functions or users. |

| **Noncompliant Code** |
| --- |
| Several threads that each invoke the do\_work() function, passing a unique number as an ID. Unfortunately, this code contains a race condition, allowing the mutex to be destroyed while it is still owned, because start\_threads() may invoke the mutex's destructor before all of the threads have exited. |
| #include <mutex>  #include <thread>    const size\_t maxThreads = 10;    void do\_work(size\_t i, std::mutex \*pm) {  std::lock\_guard<std::mutex> lk(\*pm);    // Access data protected by the lock.  }    void start\_threads() {  std::thread threads[maxThreads];  std::mutex m;    for (size\_t i = 0; i < maxThreads; ++i) {  threads[i] = std::thread(do\_work, i, &m);  }  } |

| **Compliant Code** |
| --- |
| This eliminates the race condition by extending the lifetime of the mutex. |
| #include <mutex>  #include <thread>    const size\_t maxThreads = 10;    void do\_work(size\_t i, std::mutex \*pm) {  std::lock\_guard<std::mutex> lk(\*pm);    }  std::mutex m;    void start\_threads() {  std::thread threads[maxThreads];    for (size\_t i = 0; i < maxThreads; ++i) {  threads[i] = std::thread(do\_work, i, &m);  }  } |

**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 | Probable | Hugh | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description** |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-CON50-a** | Do not destroy another thread's mutex |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | [STD-nnn-FIO] | Test and review input output for improper formats or errors |

| **Noncompliant Code** |
| --- |
| Noncompliant code example attempts to remove the trailing newline (\n) from an input line. The fgets() function is typically used to read a newline-terminated line of input from a stream. It takes a size parameter for the destination buffer and copies, at most, size - 1 characters from a stream to a character array. |
| #include <stdio.h>  #include <string.h>    enum { BUFFER\_SIZE = 1024 };    void func(void) {  char buf[BUFFER\_SIZE];    if (fgets(buf, sizeof(buf), stdin) == NULL) {  /\* Handle error \*/  }  buf[strlen(buf) - 1] = '\0';  } |

| **Compliant Code** |
| --- |
| This compliant solution uses strchr() to replace the newline character in the string if it exists |
| #include <stdio.h>  #include <string.h>    enum { BUFFER\_SIZE = 1024 };    void func(void) {  char buf[BUFFER\_SIZE];  char \*p;    if (fgets(buf, sizeof(buf), stdin)) {  p = strchr(buf, '\n');  if (p) {  \*p = '\0';  }  } else {  /\* Handle error \*/  }  } |

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

|  |
| --- |
| **Principles(s):** Keep it simple stupid. Making things complicated introduces greater chance of error  Don’t trust the user. Applying principle of least trust reduces risk and only allows the users to have access to what they need  Automation is your friend. Moving to automated audits and security checks helps remove user error. We cant forget or miss it, if it is set to operate on its own.  Threat model your software. Take a step back and plan an attack on yourself this may help uncover threats you may have overlooked |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| low | Liekly | Medium | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| **Tool** | **Version** | **Checker** | **Description** |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | **CERT\_CPP-FIO50-a** | Do not alternately input and output from a stream without an intervening flush or positioning call |
| [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.

Through the steps we must ensure best practices are followed. From the start planning for the security plan must be well thought out and organized. A strong assessment and security plan early on reduces the risk down the line. Once the design is complete refer to the SEI cert to run automated tools to test against the software. After the tests are run we can complete the build and then once again complete a verification and testing stage. The test should be a combination of automated software tests and peer code reviews. Once deployed we must utilize automated detection and intrusion tools to monitor the software. Theses tools must also be audited and collected for reference. The audited information should be reviewed by the software detection tool and verified by a person for any possible unauthorized access. If some thing is found a response plan must be ready to block attacks and if necessary shut down services. All that is left is to maintain the system with updates, reviews and new patches.

## 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 |
| --- | --- | --- | --- | --- | --- |
| DCL50-CPP | High | Probable | Medium | P12 | L1 |
| EXP50-CPP | Medium | Probable | Medium | P8 | L2 |
| INT50-CPP | Medium | Unlikely | Medium | P4 | L3 |
| CTR50-CPP | High | Likely | High | P9 | L2 |
| STR51-CPP | High | Likely | Medium | P18 | L1 |
| MEM50-CPP | High | Likely | Medium | P18 | **L1** |
| ERR50-CPP | Low | Probable | Medium | P4 | L3 |
| OOP50-CPP | Low | Unlikely | Medium | P2 | L3 |
| CON50-CPP | Medium | Probable | High | P4 | L2 |
| FIO50-CPP | Low | Liekly | Medium | P6 | L2 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## 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 | Prevent attackers from getting data when it isn’t in use |
| Encryption at flight | Encryption done while data is accessed and moving from one location to another |
| Encryption in use | Encryption of data while it is being accessed |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | A method of proving someone’s identity on the software this is important for access and permissions  Usernames and password  tokens |
| Authorization | Allows and grants appropriate action to prevent user interactions with access they are not permitted to have  Access to files  Access to crud database operation  Least privilege |
| Accounting | If a method to review authentication and authorization are not being misused  Auditing  Detection software  Intrusion software  Antivirus software  Log and record keeping |

**\***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
* Detection software(HIDS)
* Intrusion software (HIPS)
* Firewall logs
* Anti-malware logs
* Data Encryption at rest
* Data encryption while accessed
* User access 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)
* LDAP for access control
* HIDS and HIPS for instruction
* Antivirus Software
* Password policies
* Least privilege principles

# 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 |