**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 | Input should validated to prevent any unauthoried access to the software or system. Any external data by default should be considered unsafe and checked. This would include command line arguments, network request, and files passed to the program. |
| 1. Heed Compiler Warnings | Highest level compiler warnings should be used to compile code. Compiler warnings should be cleared before code can be considereded safe to operate. Use both static and dynamic testing tools to ensure that bugs and malicious code is found. |
| 1. Architect and Design for Security Policies | Software should be designed and built with security in mind. This means keeping variables, methods, and functions as secure as possible with data transmission and making best use of the security offered in the language and os that software is built for. |
| 1. Keep It Simple | KISS design should be adhered as much as possible. Using unreadable code will only make it harder to maintain and make it harder for another developer to work on in the future. This can lead to serious software issues and security flaws. |
| 1. Default Deny | All systems by default should deny users access. This will prevent a default admin account and help to prevent user id theft from compromising a system further. |
| 1. Adhere to the Principle of Least Privilege | Software should be developed with the idea of the Principle of least Privilege. By keeping accounts with as little access as needed to do there function we can help ensure the safety of the system overall. |
| 1. Sanitize Data Sent to Other Systems | Data should be checked and sanitie before it is sent to any other systems. An example would be text before its turned into SQL commands or command line arguments when they are used. |
| 1. Practice Defense in Depth | Software systems should be protected in layers. That way if one layer could be exploited the rest of the system would still be protected. |
| 1. Use Effective Quality Assurance Techniques | By utilizing good quality assurance techniques with both unit and automated testing along with both static and dynamic testing. This will make sure a good breathe of coverage is used. |
| 1. Adopt a Secure Coding Standard | Use the secure standard developed by the language decided for your work process. By utilizing a standard you ensure your code is better protected from known threats. |

### 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** | INT-30-C | Ensure that unsigned inter operations do not wrap |

| **Noncompliant Code** |
| --- |
| This example can result in an unsigned inter wrap during the addition of the usigned operands. |
| Void func ( unsigned int ui\_a, unsigned int ui\_b ){  Unsigned int usum = ui\_a + ui\_b;  /\*…\*/  } |

| **Compliant Code** |
| --- |
| The compliant solution performs a precondition test of the operands of the addition to guarantee no possible of unsigned wrap. |
| #include <limits.h>  Void func ( unsigned int ui\_a, unsigned int ui\_b ) {  Unsigned int usum;  If ( UNIT\_MAX – ui\_a < ui\_b ) {  /\* Handle error \*/  } else {  Usum = ui\_a + ui\_b;  }  /\*…\*/  } |

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

| **Principles(s):** Head Compiler Warnings. Code vulnerabilities are often pointed out by compilers. They should be fixed first before any builds to prevent unwarranted behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Interger-overflow | Fully checked |
| Axivion Bauhaus | 7.2.0 | CertC-INT30 | Implemented |
| CodeSonar | 6.1P0 | ALLOC.SIZE.ADDOFLOW  ALLOC.SIZE.IOFLOW  ALLOC.SIZE.MULOFLOW  ALLOC.SIZE.SUBUFLOW | Addition overflow of allocation size |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | DCL-51-CPP | A translation unit that includes a standard library header shall not #define or #undef names declared in any standard library header. |

| **Noncompliant Code** |
| --- |
| A common practice is to use a macro in a preprocessor conditional that guards against multiple inclusions of a header file. While this is a recommended practice, many programs use reserved names as the header guards. Such a name may clash with reserved names defined by the implementation of the C++ standard template library in its headers or with reserved names implicitly predefined by the compiler even when no C++ standard library header is included. |
| #ifndef \_MY\_HEADER\_H\_  #define \_MY\_HEADER\_H\_    // Contents of <my\_header.h>    #endif // \_MY\_HEADER\_H\_ |

| **Compliant Code** |
| --- |
| This compliant solution avoids using leading or trailing underscores in the name of the header guard. |
| #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):** Adopt a secure coding standard. By following a standard we can ensure code will be uniform and more free of errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Reserved-identifier | Partially checked |
| Axivion Bauhaus Suite | 7.20 | CertC++-DCL51 |  |
| Clang | 3.9 | -Wreserved-id-macro | The -Wreserved-id-macro flag is not enabled by default or with -Wall butis enabled with -Weverything this flag does not catch all instances of this rule. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR-31-C | Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating string. To prevent such errors, either limit copies through truncation or, preferably, ensure that the destination is of sufficient size to hold the character data to be copied and the null-termination character. (See [STR03-C. Do not inadvertently truncate a string](https://wiki.sei.cmu.edu/confluence/display/c/STR03-C.+Do+not+inadvertently+truncate+a+string).)  When strings live on the heap, this rule is a specific instance of [MEM35-C. Allocate sufficient memory for an object](https://wiki.sei.cmu.edu/confluence/display/c/MEM35-C.+Allocate+sufficient+memory+for+an+object). Because strings are represented as arrays of characters, this rule is related to both [ARR30-C. Do not form or use out-of-bounds pointers or array subscripts](https://wiki.sei.cmu.edu/confluence/display/c/ARR30-C.+Do+not+form+or+use+out-of-bounds+pointers+or+array+subscripts) and [ARR38-C. Guarantee that library functions do not form invalid pointers](https://wiki.sei.cmu.edu/confluence/display/c/ARR38-C.+Guarantee+that+library+functions+do+not+form+invalid+pointers). |

| **Noncompliant Code** |
| --- |
| This noncompliant code example demonstrates an off-by-one error. The loop copies data from src to dest. However, because the loop does not account for the null-termination character, it may be incorrectly written 1 byte past the end of dest. |
| #include <stddef.h>    void copy(**size\_t** n, **char** src[n], **char** dest[n]) {  **size\_t** i;       for (i = 0; src[i] && (i < n); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the loop termination condition is modified to account for the null-termination character that is appended to dest: |
| #include <stddef.h>    void copy(**size\_t** n, **char** src[n], **char** dest[n]) {  **size\_t** i;       for (i = 0; src[i] && (i < n - 1); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

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

| **Principles(s):** Validate Input Data, Making sure that data is input correctly and moved correctly is key to keeping data safe from outside code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Static Analysis Tool | Supported |
| Axivion Bauhaus Suite | 7.20 | CertC-STR31 | Detects calls to unsafe string fucncitons that may cause buffer overflow |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STR-02-C | String data passed to complex subsystems may contain special characters that can trigger commands or actions, resulting in a software [vulnerability](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability). As a result, it is necessary to [sanitize](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-sanitize) all string data passed to complex subsystems so that the resulting string is innocuous in the context in which it will be interpreted. |

| **Noncompliant Code** |
| --- |
| Data sanitization requires an understanding of the data being passed and the capabilities of the subsystem. John Viega and Matt Messier provide an example of an application that inputs an email address to a buffer and then uses this string as an argument in a call to system() |
| **sprintf**(buffer, "/bin/mail %s < /tmp/email", addr);  **system**(buffer); |

| **Compliant Code** |
| --- |
| It is necessary to ensure that all valid data is accepted, while potentially dangerous data is rejected or [sanitized](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-sanitize). Doing so can be difficult when valid characters or sequences of characters also have special meaning to the subsystem and may involve [validating](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-validation) the data against a grammar. In cases where there is no overlap, whitelisting can be used to eliminate dangerous characters from the data.  The whitelisting approach to data sanitization is to define a list of acceptable characters and remove any character that is not acceptable. The list of valid input values is typically a predictable, well-defined set of manageable size. This compliant solution, based on the tcp\_wrappers package written by Wietse Venema, shows the whitelisting approach: |
| static **char** ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"                           "ABCDEFGHIJKLMNOPQRSTUVWXYZ"                           "1234567890\_-.@";  **char** user\_data[] = "Bad char 1:} Bad char 2:{";  **char** \*cp = user\_data; /\* Cursor into string \*/  const **char** \*end = user\_data + **strlen**( user\_data);  for (cp += **strspn**(cp, ok\_chars); cp != end; cp += **strspn**(cp, ok\_chars)) {    \*cp = '\_';  } |

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

| **Principles(s):** Sanitize Data Sent to Other Systems. Making sure that data is checked for any malicious code before being sent other places can help keep hacks like SQL injection from happening. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Visual Stuido Code Checker | 2019 | standard | Standard compiler and checker. |
| Cpp checker | Most Recent | Static Analysis | Static Analysis tool to check for code errors and best practices. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM-52-CPP | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, 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 noexcept, 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** |
| --- |
| When 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** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  |  |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| Helix QAC | 2021.2 | C++3225, C++3226,C++3227,C++3228,C++3229,C++4632 |  |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | MSC-011-C | [Incorporate diagnostic tests using assertions](https://wiki.sei.cmu.edu/confluence/display/c/MSC11-C.+Incorporate+diagnostic+tests+using+assertions) |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses the assert() macro to verify that memory allocation succeeded. Because memory availability depends on the overall state of the system and can become exhausted at any point during a process lifetime, a robust program must be prepared to gracefully handle and recover from its exhaustion. Consequently, using the assert() macro to verify that a memory allocation succeeded would be inappropriate because doing so might lead to an abrupt termination of the process, opening the possibility of a denial-of-service attack. See also [MEM11-C. Do not assume infinite heap space](https://wiki.sei.cmu.edu/confluence/display/c/MEM11-C.+Do+not+assume+infinite+heap+space) and [void MEM32-C. Detect and handle memory allocation errors](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152159). |
| **char** \*dupstring(const **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char** \*)**malloc**(len + 1);  **assert**(NULL != dup);    **memcpy**(dup, c\_str, len + 1);    return dup;  } |

| **Compliant Code** |
| --- |
| This compliant solution demonstrates how to detect and handle possible memory exhaustion: |
| **char** \*dupstring(const **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char**\*)**malloc**(len + 1);    /\* Detect and handle memory allocation error \*/    if (NULL == dup) {        return NULL;    }    **memcpy**(dup, c\_str, len + 1);    return dup; |

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

| **Principles(s):** Use Effective Quality Assurance Techniques. IE make good use of unit and regression testing. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | P1 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Google Testing Framework | Most Recent | Test Framework | Allows the user to build robust testing for C++ programs. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR-051-CPP | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    **int** main() {    f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    **int** main() {    try {      f();    } catch (...) {      // Handle error    }  } |

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

| **Principles(s):** Practice Defense in Depth. Programs should handle all errors and check to make sure system resources aren’t compromised. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Main-function-catch-all | Partially Checked |
| Axivion Bauhaus Suite | 7.2.0 | CERTC++-ERR51 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Validation | FIO-050-CPP | [Do not alternately input and output from a file stream without an intervening positioning call](https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO50-CPP.+Do+not+alternately+input+and+output+from+a+file+stream+without+an+intervening+positioning+call) |

| **Noncompliant Code** |
| --- |
| This noncompliant code example appends data to the end of a file and then reads from the same file. However, because there is no intervening positioning call between the formatted output and input calls, the behavior is [undefined](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";    std::string str;    file >> str;  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the std::basic\_istream<T>::seekg() function is called between the output and input, eliminating the [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some 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):** Validate Input Stream. Make sure that we are checking all code input and output. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-FIO50 |  |
| Helix QAC | 2021.2 | C++4711,C++4712,C++4713 |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Orientated Programming. | OOP-50-CPP | [Do not invoke virtual functions from constructors or destructors](https://wiki.sei.cmu.edu/confluence/display/cplusplus/OOP50-CPP.+Do+not+invoke+virtual+functions+from+constructors+or+destructors) |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the base class attempts to seize and release an object's resources through calls to virtual functions from the constructor and destructor. However, the B::B() constructor calls B::seize() rather than D::seize(). Likewise, the B::~B() destructor calls B::release() rather than D::release(). |
| struct B {    B() { seize(); }    virtual ~B() { release(); }    protected:    virtual void seize();    virtual void release();  };    struct D : B {    virtual ~D() = default;    protected:    void seize() override {      B::seize();      // Get derived resources...    }      void release() override {      // Release derived resources...      B::release();    }  }; |

| **Compliant Code** |
| --- |
| In this compliant solution, the constructors and destructors call a nonvirtual, private member function (suffixed with mine) instead of calling a virtual function. The result is that each class is responsible for seizing and releasing its own resources. |
| class B {    void seize\_mine();    void release\_mine();    public:    B() { seize\_mine(); }    virtual ~B() { release\_mine(); }    protected:    virtual void seize() { seize\_mine(); }    virtual void release() { release\_mine(); }  };    class D : public B {    void seize\_mine();    void release\_mine();    public:    D() { seize\_mine(); }    virtual ~D() { release\_mine(); }    protected:    void seize() override {      B::seize();      seize\_mine();    }      void release() override {      release\_mine();      B::release();    }  }; |

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

| **Principles(s):** Keep it simple. Making sure to not overuse virtual constructors along with making sure to follow best practices will help. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Virtual-call-in-constructor | Fully Checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-OOP50 |  |
| Clang | 3.9 | Clang-analyzer-alpha.cplusplus.VirtualCall | Checked by clang-tidy |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Adopt a Secure Coding Standard | CON-050-CPP | Don’t destroy mutex while it’s locked. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example creates 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 compliant solution 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);      // Access data protected by the lock.  }    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):** Adopt a secure code standard, Heeed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C++4961,C++4962 |  |
| Klockwork | 2021.1 | CERT.CONC.MUTEX.DESTROY\_WHILE\_LOCKED |  |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

[Insert your written explanations here.]

### Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| INT-30-C | Low | Unlikely | Low | P3 | 3 |
| DCL-51-CPP | High | Likely | Medium | P18 | 1 |
| STR-31-C | High | Likely | Medium | P18 | 1 |
| STR-02-C | High | Likely | Medium | P18 | 1 |
| MEM-52-CPP | Low | Unlikely | High | P1 | 3 |
| MSC-011-C | Low | Unlikely | High | P1 | 3 |
| ERR-051-CPP | Low | Probable | Medium | P4 | 3 |
| FIO-050-CPP | Low | Likely | Medium | P6 | 2 |
| OOP-50-CPP | Low | Unlikely | Medium | P2 | 3 |
| CON-050-CPP | Medium | Probable | High | P4 | 3 |

### 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 | This will protect the data while it is stored weather that’s in a database, phone, computer, or in the cloud. There are several options for tools such as AxCrypt or InnoD8 tablespace encryption. These tools are essential to protect data from being logically stolen, pshyically stolen or otherwise breached. |
| Encryption at flight | This is the practice of sending data through secure channels. This is used via communication protocols such as TSL or SFTP. This should be used whenever possible for any company information being transferred to prevent leaks and unathorized access. |
| Encryption in use | This is the practice of keeping memory data encrypted even while in use. This can help prevent forced crashes causing memory drumps. Bitlocker is a good example so that all data on the system is encrypted so that it can’t be stolen without the key. This should be used on all company data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authenticaiton verifies a user’s identity credential. There are several ways to implement this. Via Usernames / Passwords, single sign on, biometicecs, or digital certificates. All users should be verified. This will prevent theft and unauthorized system access. |
| Authorization | Authorization defines the level of access to system items weather that be files, folders, emails , or other applications. This will prevent for instance a hacker getting ahold of a clerks creditionals and trying to access key items on the systems servers. |
| Accounting | Accounting refers to the records, or log files that detail what users have been doing. This would include when a user logs in, access files, creates a new user. This will also include when files and databases are updated. This can help prevent intercompany theft, and in the event of someone getting into a system it can help track what all they have touched. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

### Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

## Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

## Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

## Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

## Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 10/08/2021 | Updated Version | Daniel Tipton |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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