**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 | Validate input from all untrusted data sources which can eliminate many exploitable vulnerabilities. There should be suspicion of most external, untrusted data sources such as command line arguments, network interfaces, and user-controlled files. |
| 1. Heed Compiler Warnings | Code should be compiled using the highest warning level available for the complier being used and eliminate warnings by editing the code. Using static and dynamic analysis tools can help with detecting and eliminating any additional security flaws. |
| 1. Architect and Design for Security Policies | Design a software architecture with security policies in mind. The software should enforce these policies. |
| 1. Keep It Simple | Try to keep the overall design as simple as possible since the more complex designs can increase the chances that errors will occur once implemented, configured and used. There needs to be a higher effort to make sure security mechanisms are performing correctly as they become more complex. |
| 1. Default Deny | Access should be permitted only to the highest level needed for a user to perform their necessary actions. Access should be denied by default with permissions granted on a needed basis. |
| 1. Adhere to the Principle of Least Privilege | In order to prevent an attacker from having the opportunity to have higher privileged access to code, all processes should execute using the least number of privileges need to complete the request. If an elevated privilege is granted, it should only be for as little time as possible and the privilege should be revoked as soon as it is done. |
| 1. Sanitize Data Sent to Other Systems | All data that is put into complex subsystems such as command shells, relational databases, etc. should be sanitized so that attackers aren’t able to exploit the unused functionality through SQL, command or other injection attacks. |
| 1. Practice Defense in Depth | There should be multiple defense strategies so that if one layer fails, another is there to either prevent an attack or at a minimum, slow down the attacker. By having multiple layers, it can stop any exploits or at least minimize the consequences of the exploit. |
| 1. Use Effective Quality Assurance Techniques | Having good quality assurance techniques can help in identifying and removing vulnerabilities. There should be fuzz testing, penetration testing and source code audits to make sure there is an effective quality assurance technique. Even independent reviews can help with creating more secure systems. |
| 1. Adopt a Secure Coding Standard | A secure coding standard should be created and applied for the target languages and platforms being used. |

### 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] | [Make sure that integer conversions do not cause lost or misinterpreted data](https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data) |

| **Noncompliant Code** |
| --- |
| Type range errors might occur when converting from a value of an unsigned integer type to a value of a signed integer type. |
| #include <limits.h>    void func(void) {    unsigned long int u\_a = ULONG\_MAX;    signed char sc;    sc = (signed char)u\_a; /\* Cast eliminates warning \*/    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Validate ranges when converting from an unsigned type to a signed type. |
| #include <limits.h>    void func(void) {    unsigned long int u\_a = ULONG\_MAX;    signed char sc;    if (u\_a <= SCHAR\_MAX) {      sc = (signed char)u\_a;  /\* Cast eliminates warning \*/    } else {      /\* Handle error \*/    }  } |

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

| **Principles(s):** Validate Input Data: Any type of integer conversion cannot result in misinterpreted or lost data. This is especially true for any integers that come from untrusted sources. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.04 | N/A | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| CodeSonar | 7.1p0 | LANG.CAST.PC.AV LANG.CAST.PC.CONST2PTR LANG.CAST.PC.INT  LANG.CAST.COERCE LANG.CAST.VALUE  ALLOC.SIZE.TRUNC MISC.MEM.SIZE.TRUNC  LANG.MEM.TBA | Cast: arithmetic type/void pointer Conversion: integer constant to pointer Conversion: pointer/integer  Coercion alters value Cast alters value  Truncation of allocation size Truncation of size  Tainted buffer access |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Coverity | 2017.07 | NEGATIVE\_RETURNS  REVERSE\_NEGATIVE  MISRA\_CAST | Can find array accesses, loop bounds, and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior.  Can find instances where a negativity check occurs after the negative value has been used for something else.  Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | [Valid references, pointers, and iterators should be used to reference elements in a container](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR51-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+container) |

| **Noncompliant Code** |
| --- |
| For this example, pos is invalidated after the first call to insert(), and subsequent loop iterations cause [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). |
| #include <deque>    void f(const double \*items, std::size\_t count) {    std::deque<double> d;    auto pos = d.begin();    for (std::size\_t i = 0; i < count; ++i, ++pos) {      d.insert(pos, items[i] + 41.0);    }  } |

| **Compliant Code** |
| --- |
| In this compliant example, pos is assigned a valid iterator on each insertion, which helps prevent undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {    std::deque<double> d;    auto pos = d.begin();    for (std::size\_t i = 0; i < count; ++i, ++pos) {      pos = d.insert(pos, items[i] + 41.0);    }  } |

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

| **Principles(s):** Architect and Design for Security Policies: Pointers, references, and iterators share a close relationship in which it is required that referencing values be done through a valid iterator, pointer, or reference. Storing an iterator, reference, or pointer to an element within a container for any length of time comes with a risk that the underlying container may be modified such that the stored iterator, pointer, or reference becomes invalid. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | overflow\_upon\_dereference | N/A |
| CodeSonar | 7.1p0 | ALLOC.UAF | Use After Free |
| Helix QAC | 2022.3 | C++4746, C++4747, C++4748, C++4749 | N/A |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-CTR51-a | Do not modify container while iterating over it |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | [Do not attempt to create a std::string from a null pointer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR51-CPP.+Do+not+attempt+to+create+a+std%3A%3Astring+from+a+null+pointer) |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, a std::string object is created from the results of a call to std::getenv(). However, because std::getenv() returns a null pointer on failure, this code can cause [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) when the variable does not exist or may cause some other errors. |
| #include <cstdlib>  #include <string>    void f() {    std::string tmp(std::getenv("TMP"));    if (!tmp.empty()) {      // ...    }  } |

| **Compliant Code** |
| --- |
| In this solution, the results from the call to std::getenv() are being checked for a null pointer before the std::string object is constructed. This way, undefined behavior or other errors are avoided. |
| #include <cstdlib>  #include <string>    void f() {    const char \*tmpPtrVal = std::getenv("TMP");    std::string tmp(tmpPtrVal ? tmpPtrVal : "");    if (!tmp.empty()) {      // ...    }  } |

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

| **Principles(s):** Architect and Design for Security Policies: When a null pointer is dereferenced, it can cause undefined behavior and abnormal program termination. Sometimes, it can even lead to arbitrary code execution. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **assert\_failure** | N/A |
| CodeSonar | 7.1p0 | **LANG.MEM.NPD** | Null Pointer Dereference |
| Helix QAC | 2022.3 | **C++4770, C++4771, C++4772, C++4773, C++4774** | N/A |
| Parasoft C/C++test | 2022.1 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | [Do not store an already-owned pointer value in an unrelated smart pointer](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM56-CPP.+Do+not+store+an+already-owned+pointer+value+in+an+unrelated+smart+pointer) |

| **Noncompliant Code** |
| --- |
| In this code example, two different smart pointers are constructed from the same pointer value. When the local variable p2 is destroyed, it deletes the pointer value it manages. Then, when the local variable p1 is destroyed, it deletes the same pointer value, causing a [vulnerability](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-vulnerability) to appear. |
| #include <memory>    void f() {    int \*i = new int;    std::shared\_ptr<int> p1(i);    std::shared\_ptr<int> p2(i);  } |

| **Compliant Code** |
| --- |
| In this solution, the std::shared\_ptr objects are related to one another through what’s called copy construction. When the variable p2 is destroyed, the use count for the shared pointer value is decremented but still nonzero. Then, when the variable p1 is destroyed, the use count for the shared pointer value is decremented to zero, and the managed pointer is destroyed. This stops the same, already owned pointer from causing a vulnerability within the code. |
| #include <memory>    void f() {    std::shared\_ptr<int> p1 = std::make\_shared<int>();    std::shared\_ptr<int> p2(p1);  } |

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

| **Principles(s):** Architect and Design for Security Policies: Passing a pointer to a deallocated function that was not used before by the matching allocated function can result in undefined behavior. This may lead to vulnerabilities that can be exploited by attackers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | dangling\_pointer\_use | N/A |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-MEM56 | N/A |
| Helix QAC | 2022.3 | C++4721, C++4722, C++4723 | N/A |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-MEM56-a | Do not store an already-owned pointer value in an unrelated smart pointer |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | [Properly deallocate dynamically allocated resources](https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM51-CPP.+Properly+deallocate+dynamically+allocated+resources) |

| **Noncompliant Code** |
| --- |
| In this example, the variable space is passed as the expression to the placement new operator. The resulting pointer of that call is then passed to ::operator delete(), resulting in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior) due to ::operator delete() attempting to free memory that was not returned by ::operator new(). |
| #include <iostream>    struct S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {    alignas(struct S) char space[sizeof(struct S)];    S \*s1 = new (&space) S;      // ...      delete s1;  } |

| **Compliant Code** |
| --- |
| This solution removes the call to ::operator delete(), instead calling on s1's destructor. This is one of the rare occasions when invoking a destructor is allowed. |
| #include <iostream>    struct S {    S() { std::cout << "S::S()" << std::endl; }    ~S() { std::cout << "S::~S()" << std::endl; }  };    void f() {    alignas(struct S) char space[sizeof(struct S)];    S \*s1 = new (&space) S;      // ...      s1->~S();  } |

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

| **Principles(s):** Architect and Design for Security Policies: Passing a pointer to a deallocated function that was not used before by the matching allocated function can result in undefined behavior. This may lead to vulnerabilities that can be exploited by attackers. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | invalid\_dynamic\_memory\_allocation dangling\_pointer\_use | N/A |
| CodeSonar | 7.1p0 | ALLOC.FNH ALLOC.DF ALLOC.TM ALLOC.LEAK | Free non-heap variable Double free Type mismatch Leak |
| LDRA tool suite | 9.7.1 | 232 S, 236 S, 239 S, 407 S, 469 S, 470 S, 483 S, 484 S, 485 S, 64 D, 112 D | Partially Implemented |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-MEM51-a CERT\_CPP-MEM51-b CERT\_CPP-MEM51-c CERT\_CPP-MEM51-d | Use the same form in corresponding calls to new/malloc and delete/free. Always provide empty brackets ([]) for delete when deallocating arrays. Both copy constructor and copy assignment operator should be declared for classes with a nontrivial destructor. Properly deallocate dynamically allocated resources. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | [Use static assertion techniques to test expressions](https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression) in code |

| **Noncompliant Code** |
| --- |
| This noncompliant example uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly: |
| #include <assert.h>    struct timer {    unsigned char MODE;    unsigned int DATA;    unsigned int COUNT;  };    int func(void) {    assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| For assertions that only use constant expressions, a preprocessor conditional statement may be used in order to make sure it is behaving correctly. |
| struct timer {    unsigned char MODE;    unsigned int DATA;    unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** Use Effective Quality Assurance Techniques: Static assertion is a diagnostic tool to help find and remove any defects in the software that could become vulnerabilities when the code is compiled. However, not using static assertions doesn’t necessarily mean the code will be incorrect. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | N/A |
| CodeSonar | 7.1p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| ECLAIR | 1.2 | CC2.DCL03 | Fully implemented |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | [All exceptions thrown should be handled before the main() function begins](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR58-CPP.+Handle+all+exceptions+thrown+before+main%28%29+begins+executing) it’s execution |

| **Noncompliant Code** |
| --- |
| In this example, the constructor for the variable S may throw an exception that is not caught when the function globalS is constructed during program startup. |
| struct S {    S() noexcept(false);  };    static S globalS; |

| **Compliant Code** |
| --- |
| This solution makes globalS into a local variable that has static storage duration. This allows any exceptions thrown during object construction to be caught before program startup because the constructor for variable S will be executed the first time the function globalS() is called. |
| struct S {    S() noexcept(false);  };    S &globalS() {    try {      static S s;      return s;    } catch (...) {      // Handle error, perhaps by logging it and gracefully terminating the application.    }    // Unreachable.  } |

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

| **Principles(s):** Architect and Design for Security Policies: If the program throws an exception that can’t be caught, it could cause abnormal program termination. This can lead to exploitations such as denial of service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | potentially-throwing-static-initialization | Partially checked |
| CodeSonar | 7.1p0 | LANG.STRUCT.EXCP.THROW | Use of throw |
| Helix QAC | 2022.3 | C++4634, C++4636, C++4637, C++4639 | N/A |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-ERR58-a | Exceptions shall be raised only after start-up and before termination of the program |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input/Output** | [STD-008-CPP] | [Do not alternate input and output from a stream without flushing or 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 example appends data to the end of a file and then reads from the same file. But, because there is no flushing or positioning call between the output and input requests, it triggers undefined behavior. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";    std::string str;    file >> str;  } |

| **Compliant Code** |
| --- |
| In this solution, the std::basic\_istream<T>::seekg() function is called between the output and input requests. This removes the undefined behavior seen above. |
| #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):** Architect and Design for Security Policies: Constantly alternating between input and output from a stream without flushing or a positioning call can result in undefined behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO39 | N/A |
| CodeSonar | 7.1p0 | IO.IOWOP IO.OIWOP | Input After Output Without Positioning Output After Input Without Positioning |
| Helix QAC | 2022.3 | C4711, C4712, C4713  C++4711, C++4712, C++4713 | N/A |
| Parasoft C/C++test | 2022.1 | CERT\_C-FIO39-a | Do not alternately input and output from a stream without an intervening flush or positioning call |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Object Oriented Programming** | [STD-009-CPP] | [Virtual functions can’t be invoked 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 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 the solution, the constructors and destructors call a nonvirtual, private member function instead of calling a virtual function. This allows each class to handle seizing and releasing their 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):** Architect and Design for Security Policies: A virtual function can’t be invoked from a constructor/destructor that calls in the object either under construction or destruction. Making an attempt to call a derived class function from a base function can be considered dangerous because that class has not had the chance to initialize resources. In the other case, a destroyed object may have access to resources that have already been released. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | virtual-call-in-constructor invalid\_function\_pointer | Fully checked |
| CodeSonar | 7.1p0 | LANG.STRUCT.VCALL\_IN\_CTOR  LANG.STRUCT.VCALL\_IN\_DTOR | Virtual Call in Constructor  Virtual Call in Destructor |
| Helix QAC | 2022.3 | C++4260, C++4261, C++4273, C++4274, C++4275, C++4276, C++4277, C++4278, C++4279, C++4280, C++4281, C++4282 | N/A |
| Parasoft C/C++test | 2022.1 | CERT\_CPP-OOP50-a CERT\_CPP-OOP50-b CERT\_CPP-OOP50-c CERT\_CPP-OOP50-d | Avoid calling virtual functions from constructors. Avoid calling virtual functions from destructors. Do not use dynamic type of an object under construction. Do not use dynamic type of an object under destruction. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Expressions** | [STD-010-CPP] | [The order of evaluation should](https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP50-CPP.+Do+not+depend+on+the+order+of+evaluation+for+side+effects) not be depended on for side effects. |

| **Noncompliant Code** |
| --- |
| In this example, the variable i is evaluated more than once in an unsequenced manner which cause undefined behavior. |
| void f(int i, const int \*b) {    int a = i + b[++i];    // ...  } |

| **Compliant Code** |
| --- |
| The way these examples are coded means they can only be interpreted on way and there is no need to worry about the order of evaluation. |
| void f(int i, const int \*b) {    ++i;    int a = i + b[i];    // ...  }  void f(int i, const int \*b) {    int a = i + b[i + 1];    ++i;    // ...  } |

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

| **Principles(s):** **):** Architect and Design for Security Policies: Trying to change an onject in an unsequenced evaluation may cause that object to have an unexpected value which can then lead to unexpected program behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wunsequenced | Can detect simple violations of this rule where path-sensitive analysis is not required |
| CodeSonar | 7.1p0 | LANG.STRUCT.SE.DEC LANG.STRUCT.SE.INC | Side Effects in Expression with Decrement Side Effects in Expression with Increment |
| 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 |
| ECLAIR | 1.2 | CC2.EXP30 | Fully implemented |

### 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.

DevSecOps stands for development, security, and operations. It’s an approach to culture, automation, and platform design that integrates security as a shared responsibility throughout the entire development lifecycle. The idea is that security should be collaborative and not just done by one team at the end of the lifecycle. This way, security is implemented from end to end where application and infrastructure security is planned from the start. It also means automating security gates to keep the workflow from being slowed down for checks along the way. The idea is that security should be handled sooner rather than later.

### 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 | High | Medium | 2 |
| STD-002-CPP | High | Probable | High | Medium | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Likely | Medium | High | 1 |
| STD-005-CPP | High | Likely | Medium | High | 1 |
| STD-006-CPP | Low | Unlikely | High | Low | 3 |
| STD-007-CPP | Low | Likely | Low | Medium | 2 |
| STD-008-CPP | Low | Likely | Medium | Medium | 2 |
| STD-009-CPP | Low | Unlikely | Medium | Low | 3 |
| STD-010-CPP | Medium | Probable | Medium | Medium | 2 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Data at rest refers to any data residing in computer storage in any digital form. This data type is currently inactive and not moving between devices or network points. No app, service, tool or anything else is actively using this type of info. At rest encryption is a security practice of encrypting the store data to prevent unauthorized access. Data is scrambled into ciphertext and a decryption key is required to see it. This is crucial because most of this type of data contains private and valuable information. |
| Encryption at flight | This refers to any data that is being transmitted over a network. All data that goes over an internal network or the internet is vulnerable to being intercepted. Encrypting in flight means that the data is encrypted before transmission, the computer system endpoints are then authenticated and the data is decrypted and verified on arrival. |
| Encryption in use | In use encryption guarantees that sensitive data is never left unsecured, regardless of its location or lifecycle stage (at rest/in flight). The data is always encrypted so even if there is a data breach, data loss is prevented. It also analyzes data requests in real time and blocks suspicious requests. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication refers to a method of identifying a user. This is normally done by having the user input a valid username/password before access is granted into the network. Each user has unique credentials for gaining access. The AAA framework server compares entered credentials with other credentials in a stored database. If there’s a match, network access is granted but if there is not a match, network access is denied. |
| Authorization | Authorization is the process of enforcing policies where user permissions are decided. It determines what types of activities, resources or services a user is permitted to use. This usually goes hand in hand with authentication where once authentication is proven, authorization to specific processes is given. |
| Accounting | Accounting monitors the resources a user consumes during network access. This can include the amount of data sent and received during a session or logging the amount of time spent in the system. Session statistic and usage information are tracked in order to perform these tasks. This is used for authorization control, billing, trend analysis, and planning data capacity. |

**\***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 |  |
| 1.1 | 10/9/2022 | Complete Document | Ashley Santo |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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