**Green Pace Developer: Security Policy Guide Template**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

## Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

## Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

## Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | The validation of user input is the first stage in preventing errors, bugs and attacks. Checking the input to ensure it is acceptable for the task helps prevent buffer overflow/underflow. Validation is the first step to preventing SQL Injection and other attacks where a user could use malformed input to gain access to information that they shouldn’t have access to. |
| 1. Heed Compiler Warnings | The compiler is a powerful tool, it coverts our symbolic code into meaningful instructions that the computer can execute. It is also very good at finding potential security and performance issues that we may have overlooked. The compiler can “guess” if certain code segments have the potential to become problematic. This doesn’t mean our code is faulty and the program will still compile and run but there are issues that should be looked into. In short, if the compiler complains about something, you can either turn off the complaints or fix your code. Obviously, the latter is the preferred option. |
| 1. Architect and Design for Security Policies | The idea of security should start within the beginning of the SDLC. Working retail has taught me one Important lesson; customers/clients rarely know what they want but they certainly know what they DON’T want. We can follow this approach and begin to develop a security model that fits the needs of the client. On the other side, user education still needs to be addressed. If the end user isn’t educated on how to properly operate the software, there’s always going to be mishaps as the developers can’t possibly design infinite contingency plans. |
| 1. Keep It Simple | Usually, function tops form but in this case, form is better. Documentation relating to software should be as straightforward as possible. As a one of my professors stated, “no one really cares about your ability to write prose, stick to the facts.” The more simple and concise instructions are, the less likely they are open to interpretation. This also applies to code itself, unless there is a specific use-case, there’s no need to reinvent the wheel. Go with what has been established and things will work out. |
| 1. Default Deny | Everyone and everything is denied by default; only grant access when needed. This the simplest security principle. This is functionally how most unix/linux systems operate. The local user can only access their userspace. If they need to access, the system allows for temporary access that expires once it is no longer needed. Designing software with this concept in mind helps to mitigate potential threats as even a compromised user account still won’t be able to run amok. |
| 1. Adhere to the Principle of Least Privilege | Similar to default deny, users should only have the exact access they need. MySQL, a popular database system utilizes user roles to ensure data integrity. The ‘root’ user has full access but this isn’t necessary for daily tasks. Users can be granted a varying degree or read/write access, depending on their needs. An analyst only needs to be able to access the information, they need not be able to write or delete records. |
| 1. Sanitize Data Sent to Other Systems | In an ideal world, all systems would conduct their own input validation. Cleaning a program’s output, specifically if it being sent to other programs helps prevent errors and security issues. |
| 1. Practice Defense in Depth | Defense in Depth is a multi-layer approach to overall security. IT professionals will often employ antivirus, firewalls, VPNs and other such tools to protect their networks. On the software development side of things, we can implement things such as default deny and the least access privileges to |
| 1. Use Effective Quality Assurance Techniques | Use proper tools throughout the development process. Most IDEs and languages have tools that test for potential issues before code compilation. Some environments utilize dependency checkers that check libraries and other linked assets against a database of known security risks. |
| 1. Adopt a Secure Coding Standard |  |

### 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] | Use proper ranging of integer types to prevent wrapping and overflow. |

| **Noncompliant Code** |
| --- |
| The possibility of decimal truncation occurs if the value becomes too small or typecasting places the value out of range. |
| 1 unsigned long int myNum = ULONG\_MAX;  2 signed char myChar;  3 myChar = (signed myChar)myNum; /\* cast eliminates warning \*/ |

| **Compliant Code** |
| --- |
| Compare the values before doing the math, if range issues occur, handle the error |
| 1 unsigned long int myInt = ULONG\_MAX;  2 signed char myChar;  3 if (myInt <= SCHAR\_MAX) {  4 myChar = (signed char)myInt; /\* use cast to eliminate warning \*/  5 }  6 else {  7 /\* handle error condition \*/  8 } |

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

| **Principles(s):** Data validation: Input should be verified that it matches expected parameters  Data types: For arithmetic, use types of similar length and size |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Very Likely | high | 4 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2023.1 | **C2800, C2860**  **C++2800, C++2860**  **DF2801, DF2802, DF2803, DF2861, DF2862, DF2863** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2023.1 | **NUM.OVERFLOW** **CWARN.NOEFFECT.OUTOFRANGE** **NUM.OVERFLOW.DF** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **493 S, 494 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-INT32-a** **CERT\_C-INT32-b** **CERT\_C-INT32-c** | Avoid signed integer overflows Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-C++] | When working with signed/unsigned it is important to choose the correct type. Unsigned types should only be used when there is no chance the value can become negative. |

| **Noncompliant Code** |
| --- |
| The control variable is unsigned, it will eventually underflow and create an infinite loop |
| 1 char a[MAX\_ARRAY\_SIZE] = /\* initialize \*/;  2 size\_t count = /\* initialize \*/;  3  4 for (unsigned int i = count-2; i >= 0; i--) {  5 a[i] += a[i+1];  6 } |

| **Compliant Code** |
| --- |
| Using prebuilt values such as size\_t will prevent the weirdness that might occur |
| 1 char a[MAX\_ARRAY\_SIZE] = /\* initi\*/;  2 size\_t countt = /\* initi\*/;  3  4 for (size\_t i = count-2; i != SIZE\_MAX; i--) {  5 a[i] += a[i+1];  6 |

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

| **Principles(s):** Heed compiler warning / input validation. One should always listen to the warnings, they most likely are correct about possible signage issues. Validating the input, would a negative number be a bad input? Test or the possibility and ensure that it doesn’t happen. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | high | 9 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-INT30-a** **CERT\_C-INT30-b** **CERT\_C-INT30-c** | Avoid wraparounds when performing arithmetic integer operations Integer overflow or underflow in constant expression in '+', '-', '\*' operator Integer overflow or underflow in constant expression in '<<' operator |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule INT30-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint30c.html) | Checks for:   * Unsigned integer overflow * Unsigned integer constant overflow   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.25 | [**V658**](https://pvs-studio.com/en/docs/warnings/v658/)**,**[**V1012**](https://pvs-studio.com/en/docs/warnings/v1012/)**,**[**V1028**](https://pvs-studio.com/en/docs/warnings/v1028/)**,**[**V5005**](https://pvs-studio.com/en/docs/warnings/v5005/)**,**[**V5011**](https://pvs-studio.com/en/docs/warnings/v5011/) |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **unsigned overflow** | Exhaustively verified. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-C++] | Incorrect string sizes and unchecked boundaries could lead to overflow. One should never directly copy input from unbounded sources such as stdin to fixed arrays. Be aware of string encoding as this could allow a means of attack if improper coded strings are used. |

| **Noncompliant Code** |
| --- |
| If the user attempts to enter more than 19 characters, out-of-bounds write occurs and memory is potentially compromised. |
| 1 #include <iostream>  2  3 int main() {  4 char myBuf[20];  5  6 std::cin >>myBuf;  7 std::cout << "output: " << buf << '\n';  8 |

| **Compliant Code** |
| --- |
| By explicitly stating how many characters to input, we negate the issue |
| 1 #include <iostream>  2  3 int main() {  4 char myBuf[20];  5  6 std::cin.width(20);  7 std::cin >> myBuf; |

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

| **Principles(s):** Validate input and sanitize data to prevent string-based attacks and possible overflow. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | likely | Medium | 18 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule STR31-C](https://www.mathworks.com/help/bugfinder/ref/certcrulestr31c.html) | Checks for:   * Use of dangerous standard function * Missing null in string array * Buffer overflow from incorrect string format specifier * Destination buffer overflow in string manipulation * Insufficient destination buffer size   Rule partially covered. |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-STR31-a** **CERT\_C-STR31-b** **CERT\_C-STR31-c** **CERT\_C-STR31-d** **CERT\_C-STR31-e** | Avoid accessing arrays out of bounds Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Avoid using unsafe string functions which may cause buffer overflows |
| [Coverity](https://www.securecoding.cert.org/confluence/display/seccode/Coverity) | 2017.07 | **STRING\_OVERFLOW**  **BUFFER\_SIZE**  **OVERRUN**  **STRING\_SIZE** | Fully implemented |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C++] | SQL injections occur when additional information is concatenated onto the end of query strings. This information can come from command-line arguments, stored variables, user-input.  Without proper validation, attackers can inject code which could grant them access to a system, dump information or do other harmful things. |

| **Noncompliant Code** |
| --- |
| Unfiltered code can lead to an injection attack |
| SqlDataAdapter1 myCommand1 =  new SqlDataAdapter("LoginStoredProcedure '" +  Login.Text + "'", conn); |

| **Compliant Code** |
| --- |
| Dynamic SQL utilizing parameter collection will prevent appending misc data to sql strings |
| SqlDataAdapter myCommand = new SqlDataAdapter(  "SELECT au\_lname, au\_fname FROM Authors WHERE au\_id = @au\_id", conn);  SQLParameter parm = myCommand.SelectCommand.Parameters.Add("@au\_id",  SqlDbType.VarChar, 11);  Parm.Value = Login.Text; |

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

| **Principles(s):** Input validation  : Essure that input is valid and doesn’t contain anything that could be used for an attack.  Sanitize data sent to other systems: Remove any trailing characters that may get sent to an SQL server.  Default deny: access to other features such as code execution should be denied by default to prevent accidental execution of code. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | 18 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-STR31-a** **CERT\_C-STR31-b** **CERT\_C-STR31-c** **CERT\_C-STR31-d** **CERT\_C-STR31-e** | Avoid accessing arrays out of bounds Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Avoid using unsafe string functions which may cause buffer overflows |
| [Coverity](https://www.securecoding.cert.org/confluence/display/seccode/Coverity) | 2017.07 | **STRING\_OVERFLOW**  **BUFFER\_SIZE**  **OVERRUN**  **STRING\_SIZE** | Fully implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C++] | Stack Smashing or stack buffer overflow can occur when data from untrusted sources is pushed into the stack. A user could corrupt the stack in such a way that may allow for code execution. Improper use of pointers and memory allocation can increase the risk of this happening. |

| **Noncompliant Code** |
| --- |
| S is deferenced aftering being deallocated. This could allow for “write after free” |
| #include <new>  struct A {  void f();  };  void g() noexcept(false) {  A \*a = new A;  /\*\*/  delete A;  /\*\*/  A->f(); |

| **Compliant Code** |
| --- |
| If you wait until the memory is no longer needed, then it can be deleted safely without causing security issues |
| #include <new>  struct A {  void f();  };  void g() noexcept(false) {  A \*a = new A;  // ...  a->f();  delete a; |

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

| **Principles(s):** Architect for security: Those on the team should be trained on the proper use of memory allocation procedures. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | 16 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **CHECKED\_RETURN** | Finds inconsistencies in how function call return values are handled. Coverity Prevent cannot discover all violations of this recommendation, so further verification is necessary |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **leakReturnValNotUsed, ignoredReturnValue** | Return value of memory allocation function is not used.  Ignored return value from function when configuration says it must be used. See the chapter "Library configuration" in the cppcheck manual |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.EXP12** | Fully implemented |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **CHECKED\_RETURN** | Finds inconsistencies in how function call return values are handled. Coverity Prevent cannot discover all violations of this recommendation, so further verification is necessary |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C++] | Assertions test for things that should never be true. Should be evaluated with static testing |

| **Noncompliant Code** |
| --- |
| Relying solely on the assertion to handle everything is a bad idea, errors could still occur and they wouldn’t be handled |
| myErr = myGraphRoutine(a, b);  /\* No Code to handle errors \*/  ASSERT(!myErr);  \_ASSERT(!myErr); |

| **Compliant Code** |
| --- |
| Adding custom error handling helps support the assertions and ensures that nothing goes wrong |
| [myError = mytest (a, b);  /\* Code to handle errors and  reset myErr if successful \*/  ASSERT(!myErr0r);  \_ASSERT(!myErr0r); |

**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 | High | 1 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-PRE31-b** **CERT\_C-PRE31-c** **CERT\_C-PRE31-d** | Assertions should not contain assignments, increment, or decrement operators Assertions should not contain function calls nor function-like macro calls Avoid side effects in arguments to unsafe macros |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **ASSERT\_SIDE\_EFFECTS** | Partially implemented  Can detect the specific instance where assertion contains an operation/function call that may have a side effect |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-C++] | Throwing and not catching an exception is bad |

| **Noncompliant Code** |
| --- |
| No errors are caught here |
| void errorthrowing\_func() noexcept(false);  void f() {  errorthrowing\_func();  }  int main() {  f(); |

| **Compliant Code** |
| --- |
| The main() function now processes and handles errors |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  try {  f();  } catch (...) {  // Handle error here  }  } |

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

| **Principles(s):**  Keep it Simple: Keeping the code simple helps ensure that exceptions don’t go uncaught  Quality Assurance: Proper testing and assurance policies will also assign in catching missed things |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Most likely | medium | 4 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR51-a** **CERT\_CPP-ERR51-b** | Always catch exceptions Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Exception escape | [STD-008-C++] | Don’t allow exceptions escape of memory deallocation |

| **Noncompliant Code** |
| --- |
| The destructor might cause an exception and cause erratic behavior |
| 1 #include <stdexcept>  2 Class S {  3 Bool has\_error() const;  4 public:  5 ~S() noexcept(false) {  6 // Normal processing  7 if(has\_error()) {  8 throw  9 std::logic\_error("Something bad");  10 }  11 }  12 }; |

| **Compliant Code** |
| --- |
| Everything is caught |
| Class myClass {  Bad bad\_member;  public:  ~myClass()  try  {  // ...  } catch(...) {  // Catch exceptions thrown from noncompliant destructors of  // member objects or base class subobjects.  return;  }  } |

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

| **Principles(s):** Architect Design: Code should be planned and designed to minimize coding vulnerabilities. Well planned pseudo-code and UML diagrams help ensure that the proper functionality is covered.  Keep it simple – code should always be kept as simple as possible. The more complex. The more unintended things could occur. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Medium | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL57-a** **CERT\_CPP-DCL57-b** | Never allow an exception to be thrown from a destructor, deallocation, and swap Always catch exceptions |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **destructor-without-noexcept delete-without-noexcept** | Fully checked |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **destructor-without-noexcept** **delete-without-noexcept** | Fully checked |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: DCL57-CPP](https://www.mathworks.com/help/bugfinder/ref/certcdcl57cpp.html) | Checks for class destructors exiting with an exception (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-C++] | Avoid modification of namespace declaration |

| **Noncompliant Code** |
| --- |
| Adding a value x to the standard namespace (std) could cause conflicts |
| Namespace std { int  x; |

| **Compliant Code** |
| --- |
| Use a non-reserved namespace declaration and things are fine |
| Namespace notstd {  Int x; |

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

| **Principles(s):** Architect Design: Code should be planned and designed to minimize coding vulnerabilities. Well planned pseudo-code and UML diagrams help ensure that the proper functionality is covered.  Keep it simple – code should always be kept as simple as possible. The more complex. The more unintended things could occur. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.3p0 | **LANG.STRUCT.DECL.SNM** | Modification of Standard Namespaces |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | **C++3180, C++3181, C++3182** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | **CERT.DCL.STD\_NS\_MODIFIED** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-C++] | Avoid creating unnamed namespaces inside header files as this can lead to unexpected behaviors. |

| **Noncompliant Code** |
| --- |
| The variable is defined within an unnamed namespace and as a result, operates an an instance of itself |
| // test.h  #ifndef test\_HEADER\_FILE  #define test\_HEADER\_FILE  namespace  {  Int var;  }  #endif  // test.cpp  #include "test.h"  #include <iostream>  void  g() {  std::cout << "g(): " << v << std::endl;  v = 100;  }  Int main() {  Extern void f();  f(); // Prints v, sets it to 42  g(); // Prints v, sets it to 100  f();  g();  } |

| **Compliant Code** |
| --- |
| The variable is no longer inside a namespace and therefor fully visable |
| // test.h  #ifndef test\_HEADER\_FILE  #define test\_HEADER\_FILE  namespace  {  Int var;  }  #endif  // test.cpp  #include "test.h"  #include <iostream>  Int v; // Definition of global variable v  Void f() {  std::cout << "f(): " << v << std::endl;  v = 412;  }  // test2.cpp  #include "test.h"  #include <iostream>  Void h() {  std::cout << "h(): " << v << std::endl;  v = 1300;  }  Int main() {  Extern void f();  f(); // Prints v, sets it to 412  h(); // Prints v, sets it to 1300  f(); // Prints v, sets it back to 412  h(); // Prints v, sets it back to 1300 |

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

| **Principles(s): Architect Design: Code should be planned and designed to minimize coding vulnerabilities. Well planned pseudo-code and UML diagrams help ensure that the proper functionality is covered.**  **Keep it simple – code should always be kept as simple as possible. The more complex. The more unintended things could occur.** |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | 6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | unnamed-namespace-header | Fully checked |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

[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 | Very Likely | high | 4 | 2 | High |
| STD-002-CPP | Likely | high | 9 | 2 | High |
| STD-003-CPP | High | likely | Medium | 18 | 2 |
| STD-004-CPP | High | Likely | Medium | 18 | 1 |
| STD-005-CPP | High | Likely | Medium | 16 | 2 |
| STD-006-CPP | Low | Unlikely | High | 1 | 3 |
| STD-007-CPP | Low | Most likely | medium | 4 | 3 |
| STD-008-CPP | Low | Likely | Medium | 6 | 2 |
| STD-009-CPP | High | Unlikely | Medium | 6 | 2 |
| STD-010-CPP | High | Probable | Medium | 6 | 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 that is in storing should be encrypted or encoded using a passphrase or key that only the user has access to. The data will be decoded as it is being pulled from storage using the key. If the key isn’t present during the decode phase, no data is pulled from storage. |
| Encryption at flight | Data that is being transmitted from local to remote or from remote to local machine should be encoded to prevent authorized actors from accessing it. This generally done in web browsers utilizing HTTPS (Secure HTTP). Other methods might include SSH or other types of data transmission protocols that prevent data interception. |
| Encryption in use | Data stored in memory must be protected from interception. That could be accomplished by utilizing user account protection. The user-space of each user is isolated from one another. User’s can’t access the memory allocated to other users. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Identifies a user based on credentials such as ID/Password, biometrics, physical hardware, security token and so forth. Ensures that only those that need to access systems and resources can do so,. |
| Authorization | After the user has been proven, grant the access they need. Not ever user needs to have access. Default deny and least access apply. Give the use access only to what is needed to preform their specific task and restrict access to everything else. |
| Accounting | Keep a log of all authentication and authorization access and what changes are made/tasks run etc. If something breaks or there is a security breach, having a system security log helps point you in the right direction of the issue. |

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

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

### Map the Principles

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

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

* Operating system logs
* 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 | 05/23/2023 | Initial Template | Ryan Sanders |  |
| 2.0 | 06/15/2023 | Final | Ryan Sanders | [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 |