**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 | All input from unknown and untrusted sources must be validated. Input sources can be command line input, a network interface, or even reading files. Validating input can eliminate many vulnerabilities (Seacord, 2018). |
| 1. Heed Compiler Warnings | Compiler warnings are not to be ignored. The compiler should be set to high warning levels to detect the most warnings. Code should be modified to eliminate warnings, not simply suppressed. If the code is found to be correct, properly comment code to explain why the warning does not apply. Whenever possible use code checking tools to discover and correct violations (Seacord, 2018). |
| 1. Architect and Design for Security Policies | Implement a software architecture and design that will enforce security policies. As an example, if the system requires different privileges consider dividing the system into different intercommunicating subsystems, each with a different privilege level (Seacord, 2018). |
| 1. Keep It Simple | Complex designs increase the likelihood that errors will be introduced, and the level of effort required to prevent them increases (Seacord, 2018). Therefore, a simpler and less complex design is preferred whenever possible. |
| 1. Default Deny | Default to denying access and use a protection scheme that identifies the conditions where access is granted (Seacord, 2018). Base access on permission and not exclusion (Seacord, 2018). |
| 1. Adhere to the Principle of Least Privilege | Only use elevated permission access with the least amount of time required to complete the privileged task (Seacord, 2018). Doing so can prevent arbitrary code from attackers running at an elevated permission level. Every process should use the least amount of privilege necessary to complete the task (Seacord, 2018). |
| 1. Sanitize Data Sent to Other Systems | When passing data to other complex systems the data must be sanitized to prevent attackers from invoking unused functionality and creating injection attacks. The complex system does not understand the context of the call, therefore responsibility to sanitize the data falls on the system passing the data (Seacord, 2018). Complex systems can consist of command shells, relational databases, and commercial off-the-shelf components (Seacord, 2018). |
| 1. Practice Defense in Depth | To adequately manage risk, implement multiple defensive strategies. In doing so, if one defensive strategy fails, another layer can potentially prevent the exploit, or mitigate the consequences of it (Seacord, 2018). |
| 1. Use Effective Quality Assurance Techniques | Implement quality assurance techniques such as penetration testing, and source code audits to help identify and eliminate vulnerabilities (Seacord, 2018). External reviews can help to bring fresh perspective to correct invalid assumptions (Seacord, 2018). |
| 1. Adopt a Secure Coding Standard | Secure coding standards are essential for delivering high-quality software with the assurance of not exposing organizations to security incidents. Adopting a secure coding standard for your language and platform helps to achieve this goal. |

### 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** | [Guarantee that storage for character arrays has sufficient space for character data and the null terminator](https://wiki.sei.cmu.edu/confluence/display/cplusplus/VOID+STR31-CPP.+Guarantee+that+storage+for+character+arrays+has+sufficient+space+for+character+data+and+the+null+terminator) |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. |

| **Noncompliant Code** |
| --- |
| In this example a common “off by one” error occurs which results in a buffer overflow. The null terminator may be incorrectly copied one byte past the end of dest. |
| char dest[ARRAY\_SIZE];  char src[ARRAY\_SIZE];  size\_t i;  /\* ... \*/  for (i=0; src[i] && (i < sizeof(dest)); i++) {    dest[i] = src[i];  }  dest[i] = '\0';  /\* ... \*/ |

| **Compliant Code** |
| --- |
| To correct the above error, the loop termination condition must be modified to account for the null-termination character that is appended to dest. |
| char dest[ARRAY\_SIZE];  char src[ARRAY\_SIZE];  size\_t i;  /\* ... \*/  for (i=0; src[i] && (i < sizeof(dest)-1); i++) {    dest[i] = src[i];  }  dest[i] = '\0';  /\* ... \*/ |

| **Principles(s):** 3 & 10. Adopting secure coding practices and architecting secure code, such as preventing buffer overflows as shown above is crucial to adhering to this security policy. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | 145, 2845, 2846, 2847,  2848, 2849 | [Insert text.] |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Avoid Plain-Text Passwords** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Developers must take steps to prevent the leak of or unauthorized access of sensitive data. |

| **Noncompliant Code** |
| --- |
| In this small code snippet, a password is stored in memory as plain text. |
| std::string password = "mysecretpassword"; |

| **Compliant Code** |
| --- |
| To prevent storing a password in memory as plain-text it is first encrypted. |
| std::string hashedPassword = hashFunction("mysecretpassword"); |

| **Principles(s):** 8. This is a good defense in depth strategy. Should a restricted user somehow gain access to the password data, it will be encrypted and unusable. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | CP1.DCL03 | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | DF4786, DF4787, DF4788 | [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-ERR51 | [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) |
|  |  |  |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Avoid C-style Strings** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | The use of C-Style strings such as ‘char’ require explicit memory management and are prone to buffer overflows. Instead the use of standard library functions such as ‘std:string’ handle memory management automatically. |

| **Noncompliant Code** |
| --- |
| Example function that prints a string to console using C-Style strings. The function doesn’t check that the provided string is within bounds to prevent a buffer overflow. |
| void processString(char\* input) {  // Non-compliant: Using C-style string functions, potential buffer overflow risk  char buffer[50];  strcpy(buffer, "Hello, "); // Potential buffer overflow if input is too large  strcat(buffer, input); // Potential buffer overflow if input is too large  std::cout << buffer << std::endl;  } |

| **Compliant Code** |
| --- |
| Same function as above but written using standard library functions. The function uses ‘std::string’ to automatically manage memory and eliminate the risk of buffer overflow. |
| void processString(const std::string& input) {  // Compliant: Using std::string, no buffer overflow risk  std::string output = "Hello, " + input;  std::cout << output << std::endl;  } |

| **Principles(s):** 3, 4, and 10. By using std::string and taking advantage of auto memory management you simplify the code which makes it less error prone and more secure. This in turn supports adopting a secure design and security protocol. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/cplusplus/ECLAIR) | 1.2 | CP1.EXP05 | Fully implemented |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046388) | 4.10 | [S871](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-cpp.html#RSPEC-871) |  |
| [LDRA tool suite](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=99614846) | 9.7.1 | 306 S | Fully implemented |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | 3080,3082 |  |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Sanitize data passed to complex subsystems** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | 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/cplusplus/BB.+Definitions#BB.Definitions-vulnerability). |

| **Noncompliant Code** |
| --- |
| In this non-compliant example, user input is directly concatenated to the query. Since the single quote at the end of user input “John’” is not properly escaped, the malicious query will execute. |
| #include <sqlite3.h>  #include <iostream>  #include <string>  int main() {  sqlite3\* db;  char\* errMsg = nullptr;  // Open the database (replace "example.db" with the actual database name)  int rc = sqlite3\_open("example.db", &db);  if (rc) {  std::cerr << "Can't open database: " << sqlite3\_errmsg(db) << std::endl;  return rc;  }  std::string userInput = "John'; DROP TABLE users; --";  // Non-compliant: Using concatenation with untrusted data  std::string query = "SELECT \* FROM users WHERE username = '" + userInput + "'";    rc = sqlite3\_exec(db, query.c\_str(), nullptr, nullptr, &errMsg);  if (rc != SQLITE\_OK) {  std::cerr << "SQL error: " << errMsg << std::endl;  sqlite3\_free(errMsg);  }  // Clean up  sqlite3\_close(db);  return 0;  } |

| **Compliant Code** |
| --- |
| Within this compliant code, user input is bound to the query as a parameter and is not directly concatenated to the query. This will effectively prevent the malicious SQL injection of “DROP TABLE users; --;” into the query. |
| #include <sqlite3.h>  #include <iostream>  #include <string>  int main() {  sqlite3\* db;  sqlite3\_stmt\* stmt;  // Open the database (replace "example.db" with the actual database name)  int rc = sqlite3\_open("example.db", &db);  if (rc) {  std::cerr << "Can't open database: " << sqlite3\_errmsg(db) << std::endl;  return rc;  }  std::string userInput = "John'; DROP TABLE users; --";  // Compliant: Use parameterized query to avoid SQL injection  std::string query = "SELECT \* FROM users WHERE username = ?";  rc = sqlite3\_prepare\_v2(db, query.c\_str(), -1, &stmt, nullptr);  if (rc != SQLITE\_OK) {  std::cerr << "Error preparing statement: " << sqlite3\_errmsg(db) << std::endl;  sqlite3\_close(db);  return rc;  }  // Bind the parameter  sqlite3\_bind\_text(stmt, 1, userInput.c\_str(), -1, SQLITE\_TRANSIENT);  // Execute the query  while (sqlite3\_step(stmt) == SQLITE\_ROW) {  // Handle the rows  // ...  }  // Clean up  sqlite3\_finalize(stmt);  sqlite3\_close(db);  return 0;  } |

| **Principles(s):** 1, 5, and 7. Validating and sanitizing input is crucial to prevent malicious activity such as SQL injection. This also complies with the default deny principle in that your denying activities that don’t adhere to the intended functionality. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 7.4p0 | **IO.INJ.COMMAND** **IO.INJ.FMT** **IO.INJ.LDAP** **IO.INJ.LIB** **IO.INJ.SQL** **IO.UT.LIB** **IO.UT.PROC** | Command injection Format string injection LDAP injection Library injection SQL injection Untrusted Library Load Untrusted Process Creation |
| Parasoft C/C++test | 9.5 | **BD-SECURITY-{TDCMD, TDFNAMES, TDSQL}** |  |
| Klocwork | 2023.1 | [NNTS.TAINTED](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [SV.TAINTED.INJECTION](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| LDRA Tool Suite | 9.7.1 | **108 D, 109 D, 588 S** | Partially implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Do Not Access Freed Memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Evaluating or accessing a pointer that has been deallocated by memory management is undefined behavior. |

| **Noncompliant Code** |
| --- |
| S has been referenced after is has been deallocated from memory. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    delete s;    // ...    s->f();  } |

| **Compliant Code** |
| --- |
| S is properly deallocated after it is no longer needed, and not prior. |
| #include <new>    struct S {    void f();  };    void g() noexcept(false) {    S \*s = new S;    // ...    s->f();    delete s;  } |

| **Principles(s):** 4 and 10. By architecting good design principles, such as eliminating dangling pointers from code, the code is more secure by design. This in turn supports adopting a good security protocol. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | dangling\_pointer\_use |  |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-MEM50 |  |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | clang-analyzer-cplusplus.NewDelete clang-analyzer-alpha.security.ArrayBoundV2 | Checked by clang-tidy, but does not catch all violations of this rule. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | ALLOC.UAF | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Do Not Use Assertions for Input Validation** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | The use of assertions is intended for development and debugging and should not be used as a security measure. |

| **Noncompliant Code** |
| --- |
| Incorrectly use an assertion to validate data input. |
| #include <cassert>  #include <iostream>  void processFile(const char\* filename) {  // Noncompliant: Using an assertion for input validation.  assert(filename != nullptr && "Filename cannot be nullptr.");  // ... rest of the code ...  }  int main() {  const char\* sensitiveFile = "/path/to/sensitive\_file.txt";  // Noncompliant: Revealing sensitive information in assertion message.  assert(std::strlen(sensitiveFile) > 0 && "Sensitive file is empty.");  // ... rest of the code ...  } |

| **Compliant Code** |
| --- |
| Correctly use an assertion to validate assumptions. |
| #include <cassert>  void someFunction(int value) {  // Compliant: Using assertions to check valid assumptions.  assert(value >= 0 && value <= 100 && "Value should be between 0 and 100.");  // ... rest of the code ...  } |

| **Principles(s):** 3 and 9. The use of quality assurance techniques is an important step in securing software. However, the use of assertions as describe above is an example of how not to implement quality assurance. Proper design architecture will help to eliminate this type of error. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | CP1.DCL03 | Fully implemented |
| [LDRA tool suite](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=99614846) | 9.7.1 | 44 S | Enhanced Enforcement |
| [PRQA QA-C++](https://www.securecoding.cert.org/confluence/pages/viewpage.action?pageId=142409849) | 4.4 | 4090 |  |
|  |  |  |  |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Handle All Exceptions** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | When exceptions are thrown, they are processed by the nearest handler with types that match the exception thrown. |

| **Noncompliant Code** |
| --- |
| The exception thrown within throwing\_func() is not handled by f() or main(). |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| Within main() the call to f() is wrapped in a try catch to handle a thrown exception. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    try {      f();    } catch (...) {      // Handle error    }  } |

| **Principles(s):** Principles 2 and 8. Properly handling exceptions will potentially help to eliminate compiler warnings. Additionally proper try catch design patterns can help to detect potential threats and thus help implement a defense in depth strategy. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Tool | Version | Checker | Description |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | main-function-catch-all early-catch-all | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | CertC++-ERR51 |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | LANG.STRUCT.UCTCH | Unreachable Catch |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Close Files When They Are No Longer Needed** |
| --- | --- | --- |
| Input - Output | STD-008-CPP | It’s important to release the system resources used by the file when they are no longer needed. |

| **Noncompliant Code** |
| --- |
| File is not properly closed by std::terminate() alone. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    std::terminate();  } |

| **Compliant Code** |
| --- |
| File is properly close using file.close() prior to termination. |
| #include <exception>  #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }    // ...    file.close();    if (file.fail()) {      // Handle error    }    std::terminate();  } |

| **Principles(s):** 3, 4, and 10. Closing files when no longer in use will help to simplify your code and prevent errors that could arise from not releasing a resource after it is no longer required. This is good design architecture and supports the overall efforts of adopting a sound security policy. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | ALLOC.LEAK | Leak |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | DF4786, DF4787, DF4788 |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2023.1 | RH.LEAK |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-FIO51-a | Ensure resources are freed |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Range Check Element Access** |
| --- | --- | --- |
| Characters and Strings | STD-009-CPP | Index operators are unchecked and could lead to range errors. |

| **Noncompliant Code** |
| --- |
| The call to get\_index() may be greater than the number of elements available. |
| #include <string>    extern std::size\_t get\_index();    void f() {    std::string s("01234567");    s[get\_index()] = '1';  } |

| **Compliant Code** |
| --- |
| Std::basic\_string::at() throws an out or range exception if pos is >= size. The use of at() is properly wrapped in a try catch to handle the exception. |
| #include <stdexcept>  #include <string>  extern std::size\_t get\_index();    void f() {    std::string s("01234567");    try {      s.at(get\_index()) = '1';    } catch (std::out\_of\_range &) {      // Handle error    }  } |

| **Principles(s):** 3. Good design architecture such as avoiding functions that can lead to unexpected behavior such as get\_index() will help to secure the software. Properly implementing try catch to support secure code is good design. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | assert\_failure |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | LANG.MEM.BO LANG.MEM.BU LANG.MEM.TBA LANG.MEM.TO LANG.MEM.TU | Buffer overrun Buffer underrun Tainted buffer access Type overrun Type underrun |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | C++3162, C++3163, C++3164, C++3165 |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | CERT\_CPP-STR53-a | Guarantee that container indices are within the valid range |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Do Not Read Uninitialized Memory** |
| --- | --- | --- |
| Expressions | STD-010-CPP | Variables assume unexpected values when accessed before initialization. |

| **Noncompliant Code** |
| --- |
| Printing an uninitialized variable leads to unexpected behavior. |
| #include <iostream>    void f() {    int i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| Variable is properly initialized prior to use. |
| #include <iostream>    void f() {    int i = 0;    std::cout << i;  } |

| **Principles(s):** 3 and 4. Initializing data when it is declared will help to simplify the code in that it will easily eliminate the unexpected behavior associated with calling uninitialized variables. Simplifying code is good design and leads to more secure code overall because it is easier to manage. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | uninitialized-read | Partially checked |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | -Wuninitialized clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 7.4p0 | LANG.STRUCT.RPL LANG.MEM.UVAR | Return pointer to local Uninitialized variable |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2023.1 | DF726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF978 |  |

### Defense-in-Depth Illustration

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



## Project One

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

### Revise the C/C++ Standards

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

### Risk Assessment

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

### Automated Detection

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

### Automation

Provide a written explanation using the image provided.



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

Automation testing is a crucial step in the pre-production phase of the DevOps lifecycle. During pre-production typically the lifecycle begins with designing with security in mind. Building using secure practices and protocols, many of which are defined within this policy. After building, its time to verify and test that the security policies, as defined in this document, are adhered to. There are several types of automation tests, such as unit tests, integration, end-to-end, and performance. These first three steps of design, build, and verify/test represent a form of defense in depth. Keeping security in mind throughout the process and giving each step a chance to cover a potential crack in the security is important.

In a CI/CD environment speed is crucial and automation testing plays a large role in maintaining speed. By using test scripts that automatically run, it eliminates the need for manual testing which is a time-consuming endeavor. Automation testing aims to increase efficiency and removing human error which in turn can help improve test coverage.

### 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 | Likely | Medium | P18 | L1 |
| STD-002-CPP | High | Probable | Medium | P12 | L1 |
| STD-003-CPP | High | Probable | Medium | P12 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-009-CPP | High | Unlikely | Medium | P6 | L2 |
| STD-010-CPP | High | Probable | Medium | P12 | L1 |

### 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 is designed to ensure that if hackers access data on disk, it will be encrypted and unusable. Without the keys to unencrypt the data, the hackers will have to then defeat the encryption. To that end strong encryption standards should be adhered to, such as implementing AES or RSA encryption methods. A further safety measure is that the encryption keys should never be stored in the same location as the data at rest. This way if a hacker does steal a hard drive, they won’t have stolen the ability to unencrypt the data as well. |
| Encryption at flight | This represents data being transferred over a network. Secure transfer protocols should always be used. For file transfers sftp should be used over the unsecure ftp. For web servers TLS should be used for HTTPS for all connections. Wherever possible avoid using self-signed certificates. Doing so will ensure your public key is always verified by a trusted third party. |
| Encryption in use | Encryption at rest and in flight are both valuable policies however they don’t protect data that is in-use. There are several methods of accessing unencrypted data from endpoints that appear to have legitimate access to the data. In-use encryption helps eliminate this threat by keeping data encrypted through the data’s entire lifecycle. All sensitive data remains encrypted with AES-256. An additional benefit of in-use encryption is that it can detect anomalies by analyzing requests and blocking suspicious activity. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the first layer of Triple-A and it’s the way that the application identifies a user. This is the step where a user is required to login to confirm they have authorization to use the system. Authentication may be completed using username & passwords, single sign on (SSO), biometrics, digital certificates, and public key infrastructure. A good practice is to enforce two-factor authentication to as an increased step to ensure a user is who they say they are. |
| Authorization | Authorization is the granted to users by system administrators. Administrators grant each user a certain level of access that will dictate what they can, and cannot do within the system. For example, users granted a higher level of access will be given authorization to make changes to a database. Another example is only users with administration level of access will be granted the ability to add new users and determine their level of access. |
| Accounting | Accounting is the last layer of defense in a Triple-A framework. It allows administrators to audit the system, which can be helpful both for preventative maintenance but also in determining the fault after a successful breach. Accounting allows administrators to see what files were accessed by users within the system. Authentication helps with this step, because after a user is logged in to the system all tasks performed by the user are logged to that user. Which helps with the audit. |

### 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 | 08/12/2023 | Completed Template | Sean Bucklin |  |
| [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 |