**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 data that is input into a program needs to be validated so that potential security and software vulnerabilities can be avoided. Input should always be validated, and especially from any external data sources that are untrusted. It should be assumed that external data sources are suspicious, and they should be treated as such. With correct and proper use of input validation, a great deal if not all vulnerabilities. |
| 1. Heed Compiler Warnings | Warnings for the C++ compiler used should be set the highest warning level that is available. This ensures that all, if not most, compiler errors will be caught. Compiler errors found by this should be both addressed and eliminated by editing your code. Other tools beyond the compiler can be used in addition the highest compiler warning level being selected so that any other security risks can be assessed. |
| 1. Architect and Design for Security Policies | Software that is created should be done in such a way that the architecture and design both use security policies. Security should be considered before the program is created, and while it is being created to ensure that it allows for the security policies to not only be Implemented, but also enforced. |
| 1. Keep It Simple | To help ensure security, the design of a program should be as simple and small as possible. The more complex a design is the more chance there is for errors. This includes errors in the implementation, configuration, and the use of the program. If code is kept as simple and small as possible, there is far less of a chance for errors. |
| 1. Default Deny | Decisions for access in a program will be based on whether there is permission. This means that on base level, access is at fits denied. Then if there are proper conditions for access, such as credentials that have a higher level of security access, only then is access permitted. |
| 1. Adhere to the Principle of Least Privilege | All processes in the program and code should run and execute with the lowest privileges needed for the job to be completed. If elevated permissions and allowance is needed, then it should only be available until that task is completed – this reduces security risks. |
| 1. Sanitize Data Sent to Other Systems | Data should always be sanitized before it is passed to any subsystems of the program such as databases. Cyber attacks could attempt to use any functionality that is otherwise not invoked through methods such as SQL injection attacks. Sanitizing data before it enters any subsystems helps to avoid the chance of this occurring. |
| 1. Practice Defense in Depth | Defense in depth refers to using multiple layers to form a defensive barrier for the program. This means that if one layer proves to be inadequate to protect the program, there will be another layer there to protect the system. This idea capitalizes on the idea that using multiple defensive strategies will protect the system better than only using one type. Defense in depth helps to prevent one security flaw from becoming a detrimental vulnerability. |
| 1. Use Effective Quality Assurance Techniques | Effective quality assurance techniques should be used in to help to both identify and remove most if not all potential security vulnerabilities. There are multiple types of testing that can be used to help quality assurance. Examples of these testing techniques include fuzz testing, penetration testing, and source code audits. The use of independent quality assurance testers helps to ensure a more secure system. |
| 1. Adopt a Secure Coding Standard | Depending on what coding language and platform you will be using, a secure coding standard should be either developed or applied. Depending on the language or platform chosen, there are different secure coding standards. |

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

### C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

#### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CLG | All variables, functions, and return types should be defined before they are used – implicit declaration is not allowed. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/DCL31-C.+Declare+identifiers+before+using+them>

| **Noncompliant Code** |
| --- |
| Do not have a function with an implicit return type only. |
| #include <limits.h>  #include <stdio.h>    foo(void) {  return UINT\_MAX;  }    int main(void) {  long long int c = foo();  printf("%lld\n", c);  return 0;  } |

| **Compliant Code** |
| --- |
| There is an explicit definition that has a correct data type returned. |
| #include <stdlib.h>    int main(void) {  for (size\_t i = 0; i < 100; ++i) {  char \*ptr = (char \*)malloc(0x10000000);  \*ptr = 'a';  }  return 0;  } |

| **Noncompliant Code** |
| --- |
| There is no type specifier included |
| extern foo; |

| **Compliant Code** |
| --- |
| A type specifier is included with the definition |
| extern int foo; |

| **Principles(s):**  #1 Validate User Input: The input type must match what is defined for the function, and if it doesn’t there will be an error. User input must also be validated and ensured that the correct data type is entered for the function in question.  #2 Heed Compiler Warnings: All compiler warnings must be looked at and taken seriously. This could involve compiler warnings that have to do with data type declarations and return information.  #4 Keep It Simple: Implicit data types are not simple; they add a level of difficulty to be able to both understand and track. Explicit data declarations are simpler to track and to use for programs. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | type-specifier  function-return-type  implicit-function-declaration  undeclared-parameter | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL31 | Fully Implemented |
| Coverity | 2017.07 | MISRA C 2012 Rule 8.1 | Implemented |
| ECLAIR | 1.2 | CC2.DCL31 | Fully Implemented |
| Parasoft C/C++test | 2021.2 | CERT\_C-DCL31-a | All functions shall be declared before use |
| Polyspace Bug Finder | R2022a | CERT C: Rule DCL31-C | Checks for:  Types not explicitly specified  Implicit function declaration  Rule fully covered. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Initialize a variable with a value before it is read, do not allow the program to read an uninitialized variable. |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory>

| **Noncompliant Code** |
| --- |
| A value for the local variable is not initialized before the value is read, and its value is printed. |
| #include <iostream>    **void** f() {  **int** i;    std::cout << i;  } |

| **Compliant Code** |
| --- |
| A value for the local variable is initialized before it is read and used. |
| #include <iostream>    **void** f () {  **int** i = 0;    std::cout << i;  } |

| **Noncompliant Code** |
| --- |
| The memory the expression points to is not initialized |
| #include <iostream>    void f() {  int \*i = new int;  std::cout << i << ", " << \*i;  } |

| **Compliant Code** |
| --- |
| The memory is directly initialized to a value before it is printed out by the program |
| #include <iostream>    void f() {  int \*i = new int(12);  std::cout << i << ", " << \*i;  } |

| **Principles(s):**  #2 Heed Compiler Warnings: Heed the compiler warnings if a variable is uninitialized and read by the system before it is initialized.  #8 Practice Defense in Depth: Variables that are initialized before they are read by the system provide a layer of defense. This is because there are unknown consequences because we don’t know how the system will react to it. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | uninitialized-read | Partially checked |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-EXP53-a | Avoid use before initialization |
| Polyspace Bug Finder | R2022a | CERT C++: EXP53-CPP | Checks for:  Non-initialized variable  Non-initialized pointer  Rule partially covered. |
| RuleChecker | 20.10 | Uninitialized-read | Partially checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Storage for a string should be guaranteed to have sufficient storage space for both the character data and the null terminator. |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator>

| **Noncompliant Code** |
| --- |
| There is an “unbounded” input, and therefore this code will probably lead to a buffer overflow. |
| #include <iostream>    void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| The data in this code snippet is guarded for buffer overflows by using std:string. |
| #include <iostream>  #include <string>    void f() {  std::string input;  std::string stringOne, stringTwo;  std::cin >> stringOne >> stringTwo;  } |

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

| **Principles(s):**  #1 Validate Input Data: Input data needs to be validated, and there must be assurance that there is storage for both the character data and the null pointer.  #7 Sanitize Data Sent to Other Systems: If the string length is limited and has sufficient space for the null pointer, both SQL injections and buffer overflows can be prevented.  #8 Practice Defense in Depth: To help add a level of security, the string length being verified helps to ensure there is a layer of defense. If this wasn’t present hackers may use this to their advantage. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| Parasoft C/C++ test | 2021.2 | CERT\_CPP-STR50-b  CERT\_CPP-STR50-c  CERT\_CPP-STR50-e  CERT\_CPP-STR50-f  CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Do not use the 'char' buffer to store input from 'std::cin' |
| Polyspace Bug Finder | R2022a | CERT C++: STR50-CPP | 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  Rule partially covered. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-JAV] | Prevent SQL injection |

Source: https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection

| **Noncompliant Code** |
| --- |
| Code is used to authenticate a user in a system. An SQL injection is allowed by because there is an unsensitized input, “username”, into the SQL command which allows for an attacker to inject “validuser” or “1=1.” |
| class Login {  public Connection getConnection() throws SQLException {  DriverManager.registerDriver(new  com.microsoft.sqlserver.jdbc.SQLServerDriver());  String dbConnection =  PropertyManager.getProperty("db.connection");  // Can hold some value like  // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"  return DriverManager.getConnection(dbConnection);  }    String hashPassword(char[] password) {  // Create hash of password  }    public void doPrivilegedAction(String username, char[] password)  throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);    String sqlString = "SELECT \* FROM db\_user WHERE username = '"  + username +  "' AND password = '" + pwd + "'";  Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery(sqlString);    if (!rs.next()) {  throw new SecurityException(  "User name or password incorrect"  );  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  }  } |

| **Compliant Code** |
| --- |
| This code uses a placeholder for an argument, and also validates the length of a username. |
| public void doPrivilegedAction(  String username, char[] password  ) throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);    // Validate username length  if (username.length() > 8) {  // Handle error  }    String sqlString =  "select \* from db\_user where username=? and password=?";  PreparedStatement stmt = connection.prepareStatement(sqlString);  stmt.setString(1, username);  stmt.setString(2, pwd);  ResultSet rs = stmt.executeQuery();  if (!rs.next()) {  throw new SecurityException("User name or password incorrect");  }    // Authenticated; proceed  } finally {  try {  connection.close();  } catch (SQLException x) {  // Forward to handler  }  }  } |

| **Principles(s):**  #1 Validate Input Data: To help prevent SQL injection, all user input should be validated.  #3 Architect and Design for Security Policies: To help prevent SQL injection, the program should be designed to prevent it.  #7 Sanitize Data Sent to Other Systems: To help protect the system from SQL Injection, to be prepared is key. That is why statements that call for data to be sanitized both adds a layer of protection to the system as well as reduce the chance for an SQL Injection attack by hackers. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust security errors |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Parasoft Jtest | 2021.2 | CERT.IDS00.TDSQL | Protect against SQL injection |
| SonarQube | 6.7 | S2077  S3649 | Executing SQL queries is security-sensitive  SQL queries should not be vulnerable to injection attacks |
| Spotbugs | 4.6.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE  SQL\_PREPARED\_STATEMENT\_GENERATED\_FROM\_NONCONSTANT\_STRING | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Do not access memory that has been freed. |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory>

| **Noncompliant Code** |
| --- |
| A variable, s, is dereferenced after it has been deallocated. This could result in a “write-after-free”, and leaving a chance for a vulnerability that can be exploited. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

| **Compliant Code** |
| --- |
| Allocated memory for the code is not deallocated until it is no longer required for the program. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  s->f();  delete s;  } |

| **Noncompliant Code** |
| --- |
| Memory that was allocated using the “buff” object, is accessed after it was deallocated |
| #include <iostream>  #include <memory>  #include <cstring>    int main(int argc, const char \*argv[]) {  const char \*s = "";  if (argc > 1) {  enum { BufferSize = 32 };  try {  std::unique\_ptr<char[]> buff(new char[BufferSize]);  std::memset(buff.get(), 0, BufferSize);  // ...  s = std::strncpy(buff.get(), argv[1], BufferSize - 1);  } catch (std::bad\_alloc &) {  // Handle error  }  }  std::cout << s << std::endl;  } |

| **Compliant Code** |
| --- |
| The buff object is instead extended past where it is accessed previously, so that it is not accessed after it was freed |
| #include <iostream>  #include <memory>  #include <cstring>    int main(int argc, const char \*argv[]) {  std::unique\_ptr<char[]> buff;  const char \*s = "";  if (argc > 1) {  enum { BufferSize = 32 };  try {  buff.reset(new char[BufferSize]);  std::memset(buff.get(), 0, BufferSize);  // ...  s = std::strncpy(buff.get(), argv[1], BufferSize - 1);  } catch (std::bad\_alloc &) {  // Handle error  }  }  std::cout << s << std::endl; } |

| **Principles(s):**  #2 Heed Compiler Warnings: Compiler warnings should be checked for any warnings related to memory. If there is a warning, it should not be taken lightly and resolved.  #9 Use Effective Quality Assurance Techniques: To ensure that memory that has already been freed is not accessed, quality assurance should be used to check and identify if any instances of this do in fact occur and to mitigate them. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | dangling\_pointer\_use | - |
| 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. |
| Coverity | V7.5.0 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-MEM50-a | Do not use resources that have been freed |
| Polyspace Bug Finder | R2022a | CERT C++: MEM50-CPP | Checks for:  Pointer access out of bounds  Deallocation of previously deallocated pointer  Use of previously freed pointer  Rule partially covered. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CLG | Static assertions can be used to test the value of constant expressions to help find and eliminate potential software vulnerabilities. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression>

| **Noncompliant Code** |
| --- |
| In this code snippet, “assert()” is used to assert a property that involves a “memory-mapped” structure that is necessary and critical for the code to run 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** |
| --- |
| Assertions that only involve a constant expression, there can be use of a “preprocessor” conditional statement. This is compliant. |
| 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 |

| **Principles(s):**  #9 Use Effective Quality Assurance Techniques: To make the most out of the use of assertions, they should be used to help to provide quality assurance by both testing and checking the code of a program. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL03 | - |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.2p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |
| ECLAIR | 9.7.1 | CC2.DCL03 | Fully Implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle all exceptions. |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

| **Noncompliant Code** |
| --- |
| In this code snippet that is not compliant, both “f()” nor “main()” will catch exceptions that are thrown by the “throwing\_func()” function. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this code that is compliant, main entry point will handle all exceptions. Therefore, exceptions will be handled, and resources are handled successfully. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |

| **Noncompliant Code** |
| --- |
| The function, thread\_start(), does not catch all exceptions. It does not get the exceptions that are thrown by throwing\_func(). |
| #include <thread>    void throwing\_func() noexcept(false);    void thread\_start() {  throwing\_func();  }    void f() {  std::thread t(thread\_start);  t.join();  } |

| **Compliant Code** |
| --- |
| In this example, all exceptions are handled by thread\_start. |
| #include <thread>    void throwing\_func() noexcept(false);    void thread\_start(void) {  try {  throwing\_func();  } catch (...) {  // Handle error  }  }    void f() {  std::thread t(thread\_start);  t.join();  } |

| **Principles(s):**  #3 Architect and Design for Security Policies: The code should be written that helps to handle and deal with exceptions that may come up.  #9 Use Effective Quality Assurance Techniques: To help handle all exceptions, quality assurance should be done frequently. This will help to curb abnormal behavior from the code if exceptions happen that weren’t accounted for. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | main-function-catch-all  early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CerC++-ERR51 | - |
| Parasoft C/C++test | 2021.2 | 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 | R2022a | CERT C++: ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| Rule Checker | 20.10 | main-function-catch-all  early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Inputs & Outputs** | STD-008-CPP | Files should be closed when they are no longer needed. |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed>

| **Noncompliant Code** |
| --- |
| The program is terminated before the file that is open and being utilized is closed. |
| #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** |
| --- |
| The file is closed properly before the project is terminated, in noncompliant code the file used would not be terminated beforehand. |
| #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(); |

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

| **Principles(s):**  #8 Practice Defense in Depth: To help keep defense in multiple layers, files that no longer are needed should be closed. This helps to add a level of security by lessening chances of these files being accessed without permission.  #10 Adopt a Secure Coding Standard: To help keep up with the best practices of coding, all open files that are no longer needed should be closed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2022.1 | C++4786,  C++4787,  C++4788 | - |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-FIO51-a | CERT\_CPP-FIO51-a |
| Polyspace Bug Finder | R2022a | CERT C++: FIO51-CPP | Checks for resource leak (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integer | STD-009-CLG | Make sure that operations performed on signed integers do not result in an overflow. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow>

| **Noncompliant Code** |
| --- |
| This code could result in a signed integer overflow while “si\_a” and “si\_b”, which are signed, while they are added. |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Compliant codes ensure that an addition operation will not be able to overflow, or by using an extension that is available to add on to handle overflow. |
| #include <limits.h>    void f(signed int si\_a, signed int si\_b) {  signed int sum;  if (((si\_b > 0) && (si\_a > (INT\_MAX - si\_b))) ||  ((si\_b < 0) && (si\_a < (INT\_MIN - si\_b)))) {  /\* Handle error \*/  } else {  sum = si\_a + si\_b;  }  /\* ... \*/  } |

| **Noncompliant Code** |
| --- |
| This code can result in a integer overflow, specifically that may be signed. |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| There is no chance for an overflow when the subtraction is performed, it has been handled. |
| #include <limits.h>    void func(signed int si\_a, signed int si\_b) {  signed int diff;  if ((si\_b > 0 && si\_a < INT\_MIN + si\_b) ||  (si\_b < 0 && si\_a > INT\_MAX + si\_b)) {  /\* Handle error \*/  } else {  diff = si\_a - si\_b;  }    /\* ... \*/  } |

| **Noncompliant Code** |
| --- |
| Unary negation example: this can result in a signed integer overflow of the operand s\_a. It may be signed. |
| void func(signed long s\_a) {  signed long result = -s\_a;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| There is no chance for an overflow when unary negation is performed, it has been handled. |
| #include <limits.h>    void func(signed long s\_a) {  signed long result;  if (s\_a == LONG\_MIN) {  /\* Handle error \*/  } else {  result = -s\_a;  }  /\* ... \*/  } |

| **Principles(s):**  #1 ValidateInput Data: To help prevent overflow, all user input should be validated so it can’t occur.  #9 Use Effective Quality Assurance Techniques: To help avoid overflow, there should always be effective tests written that check. If quality assurance is effective, overflow can be avoided. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Integer-overflow | Fully checked |
| Parasoft C/C++test | 2021.2 | CERT\_C-INT32-a  CERT\_C-INT32-b  CERT\_C-INT32-c | Avoid integer overflows  Integer overflow or underflow in constant expression in '+', '-', '\*' operator  Integer overflow or underflow in constant expression in '<<' operator |
| Polyspace Bug Finder | R2022a | CERT C: Rule INT32-C | Checks for:  Integer overflow  Tainted division operand  Tainted modulo operand  Rule partially covered |
| PRQA QA-C | 9.7 | 2800, 2801, 2802, 2803,  2860, 2861, 2862, 2863 | Fully implemented |
| TrustInSoft Analyzer | 1.38 | signed\_overflow | Exhaustively verified (see one compliant and one non-compliant example). |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Array | STD-010-CLG | Explicitly define and specify bounds of arrays, even if they are implicitly defined. Do not do this when a character array is initialized with a string literal Is unspecified. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/ARR02-C.+Explicitly+specify+array+bounds%2C+even+if+implicitly+defined+by+an+initializer>

| **Noncompliant Code** |
| --- |
| There is an array of integers that initializes an array of integers, that contains too many elements for the array (incorrect size). |
| int a[3] = {1, 2, 3, 4}; |

| **Compliant Code** |
| --- |
| There is an explicit definition of array size, that specifies the bounds of the array. |
| int a[4] = {1, 2, 3, 4}; |

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

| **Principles(s):**  #2 Heed Compiler Warnings: If there is an error from the compiler due to arrays – make sure you fix it so it isn’t an issue.  #3 Architect and Design for Security Policies: Ensure to design your code arrays do not become an issue.  #9 Use Effective Quality Assurance Techniques: QA should ensure that arrays are correctly defined. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | array-size-global | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ARR02 | Fully implemented |
| Parasoft C/C++test | 2021.2 | CERT\_C-ARR02-a | Explicitly specify array bounds in array declarations with initializers |

### 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 and Detection**

Automation is useful in both enforcing and complying with security standards in this policy document. It is helpful to integrate security checks in all phases of production. During the pre-production phase it will be most common to use automation. In the Assess and Plan phase, you would plan what tools you want to use as well as check what threats you may be trying to plan for. In the design phase, static analysis tools will be useful as well as other built-in tools to try to catch and mitigate both errors and potential violations of the security policy. This is similar as we move to both the build and verify and test, it will be similar to use the static testing and built-in IDE tests. When moving into production, it will be more common to make use of integrity checks and rely on the use of defense-in-depth. Security should continue to be monitored and threats responded to when they come up. Automation should be used as well as checking manually – but automation is a key tool to continually monitor for issues.

### 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-CLG | Low | Unlikely | Low | Low | 1 |
| STD-002-CPP | High | Probable | Medium | High | 4 |
| STD-003-CPP | High | Likely | Medium | High | 4 |
| STD-004-JAV | High | Probable | Medium | High | 4 |
| STD-005-CPP | High | Likely | Medium | High | 5 |
| STD-006-CLG | Low | Unlikely | High | Low | 1 |
| STD-007-CPP | Low | Probable | Medium | Low | 2 |
| STD-008-CPP | Medium | Unlikely | Medium | Medium | 2 |
| STD-009-CLG | High | Likely | High | High | 4 |
| STD-010-CLG | Medium | Unlikely | Low | Medium | 3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Data at rest is the initial stage of digital data, and it is typically in a stable state. Data at rest is typically stable because it is not traveling through the system or being acted on. Encryption in rest protects this data where it is currently being stored. Encryption in rest can be used to help protect data if a device is stolen, if the PC or location where the data is stored gets hacked, and against human error. Encryption at rest varies in difficulty depending on what physical or logical data sources/storage needs to be protected, who has access to the data in question, and how much of performance and ease of access is able to be sacrificed for this protection. To protect data on hard-disk drives or USB drives, VeraCrypt, Axcrypt, or BitLocker Drive Encryption (on Windows 10) can be used. Overall, encryption at rest can happen and be successful to protect data. |
| Encryption at flight | Encryption in-transit/in-flight relates to protecting data as it is “in motion”, as in being transmitted. An example of this is when you are browsing webpages because you send a request to the server, the website accepts and processes it, and then the website is shown. Encryption in flight is useful because of threats such as “man-in-the-middle” where the attacker may alter communication so that the parties involved think they are communicating with each other. Another threat is “eavesdropping” where small pieces of data are captured from data being transmitted. Encryption in rest can be implemented using various programs and strategies. Some examples are: using Secure Sockets Layer (SSL) for web interface and traffic to transmit data and security protocols like Transport Layer Security (TLS), encryption tools for data transmitted through email like S/MIME or PGP, encrypting data before it is transmitted using Zip or RAR, and finally encrypt data that is not transmitted over the web using application-level encryption. |
| Encryption in use | Encryption in use refers to protecting data that is not only passively being stored. This could be data that is being created, edited, accessed or processed by the system. It is important to protect data in use because memory can be hacked, and the encryption keys used could be exposed. Some examples of ways to protect data in-use include using Secure Memory Encryption (SME), Total Memory Encryption (TME), and implementing End-to-End Encryption (E2EE). If a hacker was able to access data in use, a lot of sensitive information could be exposed. That is why using encryption in use is so important, it helps to protect the data that is between the two stages mentioned above. |

Source: <https://www.ryadel.com/en/data-encryption-in-transit-at-rest-definitions-best-practices-tutorial-guide/>

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication \* | Authentication is the first step in the Triple-A Framework process, it is how the system or network verifies a user’s identity. Authentication also makes sure that the user is in fact who they are claiming to be. Authentication can be confirmed by the use of username and password, biometrics (fingerprint, face recognition, etc.), S/Key, token card and server, Password Authentication Protocol (PAP), single sign-on systems (SSO), two-factor authentication, and more. Authentication for new and existing users ensures that data is protected, thought it does not completely prevent identity theft. After authentication is confirmed, the user in question then needs to be authorized to gain access to certain parts of the system. The level of data the user can access is usually tied to their authentication credentials, this dictates after authorization what they can actually access. The users that can change the database and change authorization level of users are the network and system administrators. They can also add and delete users from the system. Authentication is extremely important because it ensures that users who access the system are actually allow to access it. |
| Authorization \* | Authorization is the second step in the Triple-A Framework process. Authorization checks after a user is authenticated in the system, what the user is allowed to access in the system as well as what actions and changes they can perform. For example, a user may have the ability to type commands into the system but only be authorized to perform certain actions. A system administrator would have a greater level of authorization to perform actions then a junior network engineer. Authorization can be granted by the system administrator for users. Authorization level is linked to the authorization credentials of the users and it dictates level of user access, the files that can be accessed by users, if they can make changes to the database, and if they can add new users to the system. Authorization is necessary to ensure both security and integrity of the system. Some types of access control include mandatory access control, discretionary access control, and role-based access control. |
| Accounting \* | Accounting is the third step in the Triple-A Framework process. Accounting allows the events a user performs while accessing the system or network to be captured and logged. It basically allows for a record to be kept of all user actions, and allows for systems administrators to keep track of who accessed the system, who attempted to access the system, actions the user performed, if they were denied access to perform certain actions, and what they accessed. It is a paper trail so if there is a problem the system administrator can use this data to pinpoint who may have had access and edited or changed something they shouldn’t have. This policy allows for every action a user performs, from attempting to login and on, to be logged. It is key to be able to ensure users only are performing actions they should have the authorization to perform, and if they are able to do more they are logged and it is able to be changed. |

Source: https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting

**\***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 logsThe only item you must complete beyond this point is the Policy Version History table.

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 04/04/2022 | Draft | Nicole Penner |  |
| 3.0 | 04/23/2022 | Amended and Completed Draft | Nicole Penner |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

**References**

*ARR02-C. Explicitly specify array bounds, even if implicitly defined by an initializer - SEI CERT C Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. Retrieved April 30, 2022, from <https://wiki.sei.cmu.edu/confluence/display/c/ARR02-C.+Explicitly+specify+array+bounds%2C+even+if+implicitly+defined+by+an+initializer>

*DCL03-C. Use a static assertion to test the value of a constant expression - SEI CERT C Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. Retrieved April 30, 2022, from <https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression>

*DCL31-C. Declare identifiers before using them - SEI CERT C Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. Retrieved April 30, 2022, from <https://wiki.sei.cmu.edu/confluence/display/c/DCL31-C.+Declare+identifiers+before+using+them>

*ERR51-CPP. Handle all exceptions - SEI CERT C++ Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. Retrieved April 30, 2022, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

*EXP53-CPP. Do not read uninitialized memory - SEI CERT C++ Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory>

*FIO51-CPP. Close files when they are no longer needed - SEI CERT C++ Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. Retrieved April 30, 2022, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/FIO51-CPP.+Close+files+when+they+are+no+longer+needed>

*IDS00-J. Prevent SQL injection - SEI CERT Oracle Coding Standard for Java - Confluence*. (n.d.). Wiki.sei.cmu.edu. Retrieved April 30, 2022, from <https://wiki.sei.cmu.edu/confluence/display/java/IDS00-J.+Prevent+SQL+injection>

*INT32-C. Ensure that operations on signed integers do not result in overflow - SEI CERT C Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. Retrieved April 30, 2022, from <https://wiki.sei.cmu.edu/confluence/display/c/INT32-C.+Ensure+that+operations+on+signed+integers+do+not+result+in+overflow>

*MEM50-CPP. Do not access freed memory - SEI CERT C++ Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. Retrieved April 30, 2022, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/MEM50-CPP.+Do+not+access+freed+memory>

Mylonas, L. (2018, November 27). *AAA Security: An Introduction to Authentication, Authorisation and Accounting*. Codebots. <https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting>

Ryan. (2019, June 18). *Encryption in-transit and Encryption at-rest - Definitions and Best Practices*. Ryadel. <https://www.ryadel.com/en/data-encryption-in-transit-at-rest-definitions-best-practices-tutorial-guide/>

*STR50-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator - SEI CERT C++ Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. Retrieved April 30, 2022, from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator>