**Green Pace Developer: Security Policy**



# Green Pace Secure Development Policy

# Austin Franklin

## 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 | Validating input from untrusted data sources. Vast majority of security vulnerabilities can be properly handled by the correct implementation of user input. Always be suspicious of external data sources and assume the external data source is malicious. |
| 1. Heed Compiler Warnings | Compiler warnings are tools that are deployed by compiler designers for a reason. If there are compiler warnings, there are reasons why they exist. Simply ignoring the warnings can cause additional vulnerabilities. |
| 1. Architect and Design for Security Policies | In line with the principle of least privilege. Systems need to behave the same way. Design systems with focus on security. Separate intercommunicating systems require an appropriate level of security. |
| 1. Keep It Simple | Keep design and implementation as simple as possible. Complexities add more likelihood that errors will be introduced. If the design is simple, security will be simple as well. |
| 1. Default Deny | All access decisions need to start based on exclusion instead of inclusion. Always deny first until authorization check is done. Specific validation of permissions need to be met before granting access to any information or processes. |
| 1. Adhere to the Principle of Least Privilege | All processes only need the least amount of privileges necessary to do their task.  Reduce the time elevated privileges are needed to only the limited time the operation takes to do the task. These reduce how much time and information is presented to potential attackers. |
| 1. Sanitize Data Sent to Other Systems | Alternate to validate input data, when sending data to other systems. Send clean data. Since the system is trusted in complex systems. Sanitizing before sending to other systems prevents attacks invoking the systems own resources to use against a different system. |
| 1. Practice Defense in Depth | Manage risk using multiple layers of defense. If one layer of defense is broken, another layer of security will limit the consequences of a successful exploit. |
| 1. Use Effective Quality Assurance Techniques | Just as testing improves code quality. Regular testing of defense can improve security quality. Penetration testing, fuzz testing, and source code audits should all be independently reviewed to prevent vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Develop a clear and concise coding standard for your environment. This can be a target language and platform. Rules to follow when developing systems, as incorrectly implemented code can introduce further vulnerabilities. |

### 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** | STR50-CPP | Guarantee the storage for strings has sufficient space for the data and null terminator. Buffer overflows occur frequently when manipulating strings. Prevent errors through truncation or sufficient space. |

| **Noncompliant Code** |
| --- |
| Input is unbounded. |
| void f() {  char buf[12];  std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| Ensure data is not truncated, and guard against buffer overflows using std::string instead of bounded array. |
| #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):** Validate Input Data, assume the information passed to the program is untrusted. In this case, the input is trying to overflow the buffer. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | **MISC.MEM.NTERM**  **LANG.MEM.BO** **LANG.MEM.TO** | No space for null terminator  Buffer overrun Type overrun |
| Helix QAC | 2022.1 | **C++2835, C++2836, C++2839, C++5216** |  |
| Klocwork | 2022.1 | **NNTS.MIGHT**  **NNTS.TAINTED**  **NNTS.MUST**  **SV.UNBOUND\_STRING\_INPUT.CIN** |  |
| LDRA tool suite | 9.7.1 | **489 S, 66 X, 70 X, 71 X** | Partially implemented |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | CTR51-CPP | Use valid references, pointers, and iterators to reference elements of a container. This applies to all types of containers. Referencing values be done through a valid iterator, pointer, or reference. Any operation that invalidates a pointer or a reference should be treated as though it invalidates both pointers and references. |

| **Noncompliant Code** |
| --- |
| Pos is invalidated after the first call to insert(). Following loop iterations have undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> d;  auto pos = d.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  d.insert(pos, items[i] + 41.0);  }  } |

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

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

| **Principles(s):** Adopt a Secure Coding Standard, Keep it simple. Both relate to keeping memory management standards through tracking of pointers. Pointers need to be managed effectively and are done so by implementing a coding standard and keeping development as simple as possible. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| CTR51-CPP | High | Probable | High | **P6** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **overflow\_upon\_dereference** |  |
| CodeSonar | 6.2p0 | **ALLOC.UAF** | Use After Free |
| Helix QAC | 2022.1 | **C++4746, C++4747, C++4748, C++4749** |  |
| Klocwork | 2022.1 | **ITER.CONTAINER.MODIFIED** |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR51-CPP | Do not attempt to create a std::string from a null pointer. Passing a null pointer to this string type will result in undefined behavior because it would result in dereferencing a null pointer. |

| **Noncompliant Code** |
| --- |
| A std::string object is created from the results of a call to std::getenv(). std::getenv() returns a null pointer on failure. |
| #include <cstdlib>  #include <string>    void f() {  std::string tmp(std::getenv("TMP"));  if (!tmp.empty()) {  // ...  }  } |

| **Compliant Code** |
| --- |
| Std::getenv() is checked for null before the std::string object is constructed. |
| #include <cstdlib>  #include <string>    void f() {  const char \*tmpPtrVal = std::getenv("TMP");  std::string tmp(tmpPtrVal ? tmpPtrVal : "");  if (!tmp.empty()) {  // ...  }  } |

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

| **Principles(s):** Heed Compiler Warnings, Keep it simple, Adopt a Secure Coding Standard. This will be found by a compiler warning. But at the same time a coding standard needs to be implemented to prevent this before it reaches the compiler. Secure coding practices is an effective tool. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| STR51-CPP | High | Likely | Medium | **P18** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **assert\_failure** |  |
| Helix QAC | 2022.1 | **C++4770, C++4771, C++4772, C++4773, C++4774** |  |
| Klocwork | 2022.1 | [NPD.CHECK.CALL.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.CHECK.CALL.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.CHECK.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)  [NPD.CHECK.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)  [NPD.CONST.CALL](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.CONST.DEREF](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.FUNC.CALL.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.FUNC.CALL.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.FUNC.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.FUNC.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)  [NPD.GEN.CALL.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.GEN.CALL.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.GEN.MIGHT](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [NPD.GEN.MUST](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/)  [RNPD.CALL](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) [RNPD.DEREF](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| Parasoft C/C++test | 2021.2 | **CERT\_CPP-STR51-a** | Avoid null pointer dereferencing |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | IDS00-J | Prevent SQL injection. Prevents untrusted data from obtaining unauthorized access. |

| **Noncompliant Code** |
| --- |
| Modifies the doPrivligedAction method to use a preparedstatement instead of java.sql.Statement. However, unsanitized input argument username is incorporated into the prepared statement. |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.Statement;    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;        PreparedStatement stmt = connection.prepareStatement(sqlString);          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        }      }    }  } |

| **Compliant Code** |
| --- |
| The symbol ? is used as a placeholder for the argument. Validates the length of the username argument, preventing the user from submitting long 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      }    }  } |

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

| **Principles(s):** Validate Input Data, Sanitize Data Sent to Other Systems. All input being passed to databases needs to be parameterized to prevent SQL injection. Treat all data as untreated. This applies to all components of a complex system. Especially components that interact with databases. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| IDS00-J | High | Probable | Medium | **P12** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | **Tainting Checker** | Trust and security errors (see Chapter 8) |
| CodeSonar | 6.2p0 | **JAVA.IO.INJ.SQL** | SQL Injection (Java) |
| Coverity | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_**  **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented |
| Findbugs | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM50-CPP | Do not access freed memory. Using memory that has been deallocated by a memory management function is undefined behavior. |

| **Noncompliant Code** |
| --- |
| S is dereferenced after it has been deallocated. If this access results in a write-after-free, the vulnerability can be exploited to run arbitrary code with the permissions of the vulnerable process. |
| #include <new>    struct S {  void f();  };    void g() noexcept(false) {  S \*s = new S;  // ...  delete s;  // ...  s->f();  } |

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

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

| **Principles(s):** Use Effective Quality Assurance Techniques, Adopt a Secure Coding Standard, Heed compiler warnings. Effective quality Assurance Techniques would help prevent this issue. QA techniques would help identify this issue after being deployed. Improving code quality. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| MEM50-CPP | High | Likely | Medium | **P18** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **dangling\_pointer\_use** |  |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-MEM50** |  |
| 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 | 6.2p0 | **ALLOC.UAF** | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | DCL03-C | Use a static assertion to test the value of a constant expression. |

| **Noncompliant Code** |
| --- |
| Uses assert() to assert a property concerning a memory-mapped structure. |
| #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** |
| --- |
| Preprocessor conditional statement is used for asserting constant expressions. |
| struct timer {    unsigned char MODE;    unsigned int DATA;    unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))    #error "Structure must not have any padding"  #endif |

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

| **Principles(s):** Use quality assurance techniques. Using static assertions helps improve code quality and can audit code to prevent vulnerabilities. Asserting constant expressions requires static assertions. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| DCL03-C | Low | Unlikely | High | **P1** |

**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 |
| Compass/ROSE |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | Handle all exceptions. The default terminate handler calls abort(), which abnormally terminates the process. If the process does not unwind the stack, external resources may be left in an indeterminate state. |

| **Noncompliant Code** |
| --- |
| Neither functions catch exceptions thrown by throwing\_func(). Std::terminate() is called. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    f();  } |

| **Compliant Code** |
| --- |
| The main entry point handles all exceptions, ensures stack is unwound up to the main() function |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    int main() {    try {      f();    } catch (...) {      // Handle error    }  } |

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

| **Principles(s):** Adopt a secure coding standard, Use effective QA Techniques. Creating standards of secure coding makes code consistent and developers follow the same implementations. Effective testing can help catch unexpected errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| ERR51-CPP | Low | Probable | Medium | **P4** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **main-function-catch-all early-catch-all** | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-ERR51** |  |
| Helix QAC | 2022.1 | **C++4035, C++4036, C++4037** |  |
| Klocwork | 2022.1 | **MISRA.CATCH.ALL** |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Expressions | EXP34-C | Do not dereference null pointers. Doing so results in undefined behavior as well as abnormal program termination. |

| **Noncompliant Code** |
| --- |
| Libpng implements its own wrapper to malloc. Returns a null pointer on error or being passed a 0-byte-length requirement. If length has value of -1, the addition yields 0 and png\_malloc returns a nullpointer. Resulting in user-defined data overwriting memory starting at address 0x0. |
| #include <png.h> /\* From libpng \*/  #include <string.h>    void func(png\_structp png\_ptr, int length, const void \*user\_data) {  png\_charp chunkdata;  chunkdata = (png\_charp)png\_malloc(png\_ptr, length + 1);  /\* ... \*/  memcpy(chunkdata, user\_data, length);  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Ensures that png\_malloc() is not null. Also ensures negative numbers are not passed to func() with the use of unsigned integer. |
| #include <png.h> /\* From libpng \*/  #include <string.h>    void func(png\_structp png\_ptr, size\_t length, const void \*user\_data) {  png\_charp chunkdata;  if (length == SIZE\_MAX) {  /\* Handle error \*/  }  if (NULL == user\_data) {  /\* Handle error \*/  }  chunkdata = (png\_charp)png\_malloc(png\_ptr, length + 1);  if (NULL == chunkdata) {  /\* Handle error \*/  }  /\* ... \*/  memcpy(chunkdata, user\_data, length);  /\* ... \*/    } |

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

| **Principles(s):** Architect and Design for Security Policies. Ensure that security and handling of vulnerabilities is of high importance. Defensive coding should be followed to maintain security policies. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| EXP34-C | High | Likely | Medium | **P18** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | **null-dereferencing** | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-EXP34** |  |
| CodeSonar | 6.2p0 | **LANG.MEM.NPD LANG.STRUCT.NTAD LANG.STRUCT.UPD** | Null pointer dereference Null test after dereference Unchecked parameter dereference |
| Compass/ROSE |  |  | Can detect violations of this rule. In particular, ROSE ensures that any pointer returned by malloc(), calloc(), or realloc() is first checked for NULL before being used (otherwise, it is free()-ed). ROSE does not handle cases where an allocation is assigned to an [lvalue](#lvalue) that is not a variable (such as a struct member or C++ function call returning a reference) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declarations | DCL58-CPP | Do not modify the standard namespaces. Reduce the likelihood of conflicting identifiers. If new declarations are introduced into the standard namespace, this produces undefined behavior. |

| **Noncompliant Code** |
| --- |
| X is added to the namespace std, resulting in undefined behavior. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| Place x into a separate non collision namespace to prevent issues with identifiers. |
| namespace nonstd {  int x;  } |

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

| **Principles(s):** Adopt a secure coding standard, Keep it simple. Modifying the standard namespace causes undefined behavior. Keeping it simple is not modifying standard default namespaces which by itself is keeping a coding standard. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| DCL58-CPP | High | Unlikely | Medium | **P6** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-DCL58** |  |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2022.1 | **C++3180, C++3181, C++3182** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2022.1 | [**CERT.DCL.STD\_NS\_MODIFIED**](https://support.roguewave.com/documentation/klocwork/en/current/certcandcsecurecodingstandardidsmappedtoklocworkcandccheckers/) |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2021.2 | **CERT\_CPP-DCL58-a** | Do not modify the standard namespaces 'std' and 'posix' |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| String Literals | STR30-C | Do not attempt to modify string literals. At compile time, string literals are used to create an array of static storage duration of sufficient length. Modifying string literals results in undefined behavior and are read-only memory. |

| **Noncompliant Code** |
| --- |
| The char pointer str is initialized to the address of the string literal. |
| char \*str = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| A string literal specifies the initial values of characters in an array as well as the size of the array. This code creates a copy of the string literal to a string. This new string can be modified safetly. |
| char str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** Heed compiler warnings. Modifying string literals will be shown as a warning. Trust the compiler as string literals should not be modified. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| STR30-C | Low | Likely | Low | **P9** |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | **string-literal-modfication** **write-to-string-literal** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-STR30** | Fully implemented |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect simple violations of this rule |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **PW** | Deprecates conversion from a string literal to "char \*" |

### 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.

The standards in this policy need to be enforced mainly within the preproduction cycle. Starting at the Assess and plan section. Defining a new standard policy would be deployed. The DevOps pipeline with continuous integration and continuous deployment would start the iteration on the new standards. Designing best practices and tests related to the new vulnerabilities. Building and verifying through further testing. Which is then deployed and fixes applied then goes through the production phase. The new standards need to start at the automated testing. Since SecDevOps need automation, as it is a core component. But the new standards also don’t need to be in a single cycle. These development lifecycles are not using waterfall methods, but iterative development.

The tests need to be designed carefully to meet the requirements of the risks based on severity. And automatically checking dependencies used within the application can greatly reduce the number of vulnerabilities. When in production, the logging needs to be automated and monitored to check for potential unauthorized access and breach attempts.

### 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 |
| --- | --- | --- | --- | --- | --- |
| STR50-CPP | High | Unlikely | Medium | High | 2 |
| CTR51-CPP | High | Probable | High | **P6** | **L2** |
| STR51-CPP | High | Likely | Medium | **P18** | **L1** |
| IDS00-J | High | Probable | Medium | **P12** | **L1** |
| MEM50-CPP | High | Likely | Medium | **P18** | **L1** |
| DCL03-C | Low | Unlikely | High | **P1** | **L3** |
| ERR51-CPP | Low | Probable | Medium | **P4** | **L3** |
| EXP34-C | High | Likely | Medium | **P18** | **L1** |
| DCL58-CPP | High | Unlikely | Medium | **P6** | **L2** |
| STR30-C | Low | Likely | Low | **P9** | **L2** |
|  |  |  |  |  |  |

### 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 | Encryption at rest is referring to encrypting data that is currently not being used. This is a typical example for web applications that have authentication information. Storing of username and passwords. These should not be stored in plain text. Same goes for any payment information stored in rest. All sensitive data should be encrypted while in storage. |
| Encryption at flight | Encryption in flight or in transit is encrypting data before transmission. The computer system endpoints are authenticated, then the data is decrypted on arrival. It is used in practice to prevent other systems from seeing communication between two devices. Typically any information that is sensitive being passed between clients or servers. It should be applied at all time across networks and especially in public places. Servers use SSL to handle encrypting traffic. |
| Encryption in use | Encryption in use is when the data is currently opened by one or more applications. To be used either by the application or accessed by the user of the application. Some cases are when the content is decrypted after a user enters a password for an encrypted file. In use encryption can be protected through identity management tools like two factor authentication. Verifying that the user says who they say they are. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the method used to identify the user. Typically having to enter a username and password before being granted to the user. It is used to validate that the user is who they say they are. It provides the gateway to access an application, network, or system. While also blocking users who do not have valid credentials. |
| Authorization | Authorization is checking what permissions to grant for the user after they have authenticated to the system. For example, some administrators may have full access to make changes to databases, add new users. While other users may only have access to specific files in a system. Authorization drives user level access allowing principle of least privilege. |
| Accounting | Accounting is logging and accountability for the system and data. It audits all valuable information in a system. This could be shown as which users last modified a file or system components. Or simply logging all transactions from users. Tracking unauthorized access can help identify access attempts. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 3/15/2022 | Milestone Three – Coding Standards | Austin Franklin |  |
| 1.2 | 4/6/2022 | Project One – Security Policy | Austin Franklin |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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