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



Green Pace Secure Development Policy

Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

# Overview

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

# Purpose

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

# Scope

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

# Module Three Milestone

## Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validate input from all untrusted data sources. Assume all input is malicious. |
| 1. Heed Compiler Warnings | Use the highest available warning level for your compiler. Supplement this with static and dynamic analysis tools to detect and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | Create, design, and implement software that enforces security policies. |
| 1. Keep It Simple | Keep the design simple and small which increases strong coding standards and decreases design errors. |
| 1. Default Deny | Create protection schemes that grant access under certain conditions, rather than exclusions. |
| 1. Adhere to the Principle of Least Privilege | Any elevated permission should only be given a certain amount of time to the access of that privileged task. |
| 1. Sanitize Data Sent to Other Systems | Prevention against SQL, command, and other injection attacks. This sanitizes the data before accessing the subsystem. |
| 1. Practice Defense in Depth | Have a security strategy that consists of multiple layers. Make sure runtime environments are as secure as your code. |
| 1. Use Effective Quality Assurance Techniques | Identifies and eliminates vulnerabilities. Use various assurance programs such as code reviews and penetration testing. |
| 1. Adopt a Secure Coding Standard | Follow the Open Web Application Security Project(OWASP) for guidelines on adhering to these coding standards. |

## C/C++ Ten Coding Standards

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

### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP  [DLC30-C] | Declare objects with appropriate storage durations. |

| **Noncompliant Code** |
| --- |
| The function init\_array() returns a pointer to a character array with automatic storage duration, which is accessible to the caller. |
| **char** \*init\_array(**void**) {  **char** array[10];    /\* Initialize array \*/  **return** array;  } |

| **Compliant Code** |
| --- |
| If it is necessary for p to be defined with static storage duration but c\_str with a more limited duration, then p can be set to NULL before c\_str is destroyed. This practice prevents p from taking on an indeterminate value, although any references to p must check for NULL. |
| **const** **char** \*p;  **void** is\_this\_OK(**void**) {  **const** **char** c\_str[] = "Everything OK?";    p = c\_str;    /\* ... \*/    p = NULL;  } |

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

|  |
| --- |
| **Principles(s):** Every object has a storage duration that determines its lifetime. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Pointered-deallocation  Return-reference-local | Fully checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC-DCL30 | Fully implemented |
| CodeSonar | 6.0p0 | LANG.STRUCT.RPL | Returns pointer to local |
| Coverity | 2017.07 | RETURN\_LOCAL | Finds many instances where a function will return a pointer to a local stack variable. Coverity Prevent cannot discover all violations of this rule, so further verification is necessary |
| LDRA tool suite | 9.7.1 | 42 D, 77 D, 71 S, 565 S | Enhanced Enforcement |
| Parasoft C/C++test | 2020.2 | CERT\_C-DCL30-a  CERT\_C-DCL30-b | The address of an object with automatic storage shall not be returned from a function. The address of an object with automatic storage shall not be assigned to another object that may persist after the first object has ceased to exist |
| PC-lint Plus | 1.4 | 604, 674, 733, 789 | Partially supported |
| Polyspace Bug Finder | R2020a | CERT C: RULE DCL30-C | Checks for pointer or reference to stack variable leaving scope |
| PRQA QA-C | 9.7 | 3217, 3225, 3230, 4140 | Partially implemented |
| RuleChecker | 20.10 | Return-reference-local | Partially checked |
| TrustInSoft Analyzer | 1.38 | Dangling\_pointer | Exhaustively detects undefined behavior |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP  [INT32-C] | Ensure that operations on signed integers do not result in overflow. |

| **Noncompliant Code** |
| --- |
| This example can result in a signed integer overflow during the addition of the signed operands si\_a and si\_b. |
| **void** func(**signed** **int** si\_a, **signed** **int** si\_b) {  **signed** **int** sum = si\_a + si\_b;    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| This ensures that the addition operation cannot overflow, regardless of representation. |
| #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;    }    /\* ... \*/  } |

|  |
| --- |
| **Principles(s):** It is important to ensure that operations on signed integers do not result in overflow. Of particular importance are operations on signed integer values that originate from a tainted source and are used as:   * Integer operands of any pointer arithmetic, including array indexing * The assignment expression for the declaration of s variable length array * Function arguments of type size\_t or rsize\_t |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | High | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astée | 20.10 | Integer-overflow | Fully checked |
| TrustInSoft Analyzer | 1.38 | Signed\_overflow | Exhaustively verified |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Parasoft  C/C++test | 2020.2 | 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 |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP  [STR52-CPP] | Use valid references, pointers, and iterators to reference elements of a basic\_string. |

| **Noncompliant Code** |
| --- |
| This example copies input into a std::string, replacing semicolon(;) characters with spaces. |
| #include <string>    **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      email.insert(loc, \*i != ';' ? \*i : ' ');    }  } |

| **Compliant Code** |
| --- |
| The value of the iterator loc is updated as a result of each call to insert() so that the invalidate iterator in never accessed. The update iterator is then incremented at the end of the loop. |
| |  | | --- | | #include <string>    **void** f(**const** std::string &input) {    std::string email;      // Copy input into email converting ";" to " "    std::string::iterator loc = email.begin();  **for** (auto i = input.begin(), e = input.end(); i != e; ++i, ++loc) {      loc = email.insert(loc, \*i != ';' ? \*i : ' ');    }  } | |

|  |
| --- |
| **Principles(s):** Since std::basic\_string is a container of characters; this rule is a specific instance of CTR51-CPP. Use valid references, pointers, and iterators to reference elements of a container. As a container, it supports iterators just like other containers in the Standard Template Library. However, the std::basic\_string template class has unusual invalidation semantics. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-STR52-a | Use valid references, pointers, and iterators to reference elements of a basic\_string |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP  [IDS00-J] | Prevent SQL injection. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows JDBC code to authenticate a user to a system. The password is passed as a char array, the database connection is created, and then the passwords are hashed.  This code example permits an SQL injection attack by incorporating the unsanitized input argument username into the SQL command, allowing an attacker to inject validuser' OR '1'='1. The password argument cannot be used to attack this program because it is passed to the hashPassword() function, which also [sanitizes](https://wiki.sei.cmu.edu/confluence/display/java/Rule+BB.+Glossary#RuleBB.Glossary-sanitize) the input. |
| **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 + "'";        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 compliant solution uses a parametric query with a ? character as a placeholder for the argument. This code also validates the length of the username argument, preventing an attacker from submitting an arbitrarily long user name. |
| **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):** SQL injection vulnerabilities arise in applications where elements of a SQL query originate from an untrusted source. Without precautions, the untrusted data may maliciously alter the query, resulting in information leaks or data modification. The primary means of preventing SQL injection are sanitization and validation, which are typically implemented parameterized queries and stored procedures. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| 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 | Implemented |
| Fortify | 1.0 | HTTP\_Response\_Splitting  SQL\_Injection\_Persistence  SQL\_Injection | Implemented |
| Klocwork |  | SV.DATA.BOUND  SV.DATA.DB  SV.HTTP\_SPLIT  SV.PATH  SV.PATH.INJ  SV.SQL | Implemented |
| Parasoft Jtest | 2020.2 | BD-SECURITY-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 |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP  [MEM52-CPP] | Detect and handle memory allocation errors. |

| **Noncompliant Code** |
| --- |
| In this example, an array of int is created using ::operator new[] (std::size\_t) and the results of the allocation are not checked. The function is marked as noexcept, so the caller assumes this function does not throw any exceptions. |
| #include <cstring>    **void** f(const **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = new **int**[size];    std::**memcpy**(copy, array, size \* sizeof(\*copy));    // ...  **delete** [] copy;  } |

| **Compliant Code** |
| --- |
| When using std::nothrow, the new operator returns either a null pointer or a pointer to the allocated space. Always test the returned pointer to ensure it is not nullptr before referencing the pointer. This compliant solution handles error condition when the returned pointer is nullptr. |
| #include <cstring>  #include <new>    **void** f(**const** **int** \*array, std::**size\_t** size) noexcept {  **int** \*copy = **new** (std::**nothrow**) **int**[size];  **if** (!copy) {      // Handle error      return;    }    std::**memcpy**(copy, array, size \* **sizeof**(\*copy));    // ...  **delete** [] copy;  } |

|  |
| --- |
| **Principles(s):** When using the nonthrowing form, it is imperative to check that the return value is not nullptr before accessing the resulting pointer. When using either form, be sure to comply with ERR50-CPP. Do not abruptly terminate the program. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-MEM52-a  CERT\_CPP-MEM52-b | Check the return value of new.  Do not allocate resources in function argument list because the order of evaluation of a function’s parameters is undefined |
| Polyspace Bug Finder | R2020a | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP  [ERR06-C] | Understand the termination behavior of assert() and abort(). |

| **Noncompliant Code** |
| --- |
| This example defines a function that is called before the program exits to clean up. |
| void cleanup(void) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    **int** main(void) {    if (**atexit**(cleanup) != 0) {      /\* Handle error \*/    }      /\* ... \*/    **assert**(/\* Something bad didn't happen \*/);      /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The call to assert() is replaced with an if statement that calls exit() to ensure that the proper termination routines are run. |
| void cleanup(void) {    /\* Delete temporary files, restore consistent state, etc. \*/  }    **int** main(void) {    if (**atexit**(cleanup) != 0) {      /\* Handle error \*/    }      /\* ... \*/      if (/\* Something bad happened \*/) {  **exit**(EXIT\_FAILURE);    }      /\* ... \*/  } |

|  |
| --- |
| **Principles(s):** The assert macro puts diagnostic tests into programs; it expands to a void expression. When executed, if expression is false, the assert macro writes information about the particular call that failed on the standard error stream in an implementation-defined format. It then calls the abort function. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced enforcement |
| Parasoft C/C++test | 2020.2 | CERT\_C-ERR06-a | Do not use assertions |
| PC-lint Plus | 1.4 | 586 | Fully supported |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP  [ERR51-CPP] | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| Neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, 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, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {    throwing\_func();  }    **int** main() {    try {      f();    } catch (...) {      // Handle error    }  } |

|  |
| --- |
| **Principles(s):** All exceptions thrown by an application must be caught by a matching exception handler. Even if the exception cannot be gracefully recovered from using the matching exception handler ensures that the stack will be properly unwound and provides an opportunity to gracefully manage external resources before terminating the process. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++test | 2020.2 | 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 | R2020a | CERT C++: ERR51-CPP | Checks for unhandled exceptions |
| RuleChecker | 20.10 | Main-function-catch-all  Early-catch-all | Partially checked |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Error Handling** | STD-00-CPP  [ERR58-CPP] | Handle all exceptions thrown before main() begins executing. |

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

| **Compliant Code** |
| --- |
| This solution makes globalS into a local variable with static storage duration, allowing any exceptions thrown during object construction to be caught because the constructor for S will be executed the first time the function globalS() is called rather that at program startup. This solution does require the programmer to modify source code so that previous uses of globalS are replaced by a function call to globalS(). |
| **struct** S {    S() noexcept(**false**);  };    S &globalS() {  **try** {  **static** S s;  **return** s;    } **catch** (...) {      // Handle error, perhaps by logging it and gracefully terminating the application.    }    // Unreachable.  } |

|  |
| --- |
| **Principles(s):** If an uncaught exception is thrown before main() is executed, or if an uncaught exception is thrown after main() has finished executing, there are no further opportunities to handle the exception and it results in implementation-defined behavior. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Potentially-throwing-static-initialization | Partially checked |
| Clang | 3.9 | Cert-err58-cpp | Checked by clang-tidy |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-ERR58-a | Exceptions shall be raised only after start-up and before termination of the program |
| RuleChecker | 20.10 | Potentially-throwing-static-initialization | Partially checked |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input/Output** | STD-009-CPP  [FIO06-C] | Create files with appropriate access permissions |

| **Noncompliant Code** |
| --- |
| Using the POSIX open() function to create a file but failing to provide access permissions for that file may cause the file to be created with overly permissive access permissions. This omission has been known to lead to vulnerabilities. |
| **char** \*file\_name;  **int** fd;  /\* Initialize file\_name \*/  fd = open(file\_name, O\_CREAT | O\_WRONLY);  /\* Access permissions were missing \*/  **if** (fd == -1){  /\* Handle error \*/  } |

| **Compliant Code** |
| --- |
| Access permissions for the newly created file should be specified in the third argument to open(). Again, the permissions are modified by the value of umask(). |
| **char** \*file\_name;  **int** file\_access\_permissions;  /\* Initialize file\_name and file\_access\_permissions \*/  **int** fd = open(  file\_name,  O\_CREAT | O\_WRONLY,  file\_access\_permissions  );  **if** (fd == -1){  /\* Handle error \*/  } |

|  |
| --- |
| **Principles(s):** Creating a file with insufficiently restrictive access permissions may allow an unprivileged user to access that file. Although access permissions are heavily dependent on the file system, many file-creation functions provide mechanisms to set access permissions. When these functions are used to create files, appropriate access permissions should be specified to prevent unintended access. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | (customization) | CodeSonar’s custom checking infrastructure allows users to implement checks such as the following.   * A check for all uses of fopen(). * A check for calls to open() with only two arguments. * A check for calls to open() where the third argument does not satisfy some specified requirement. |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced Enforcement |
| PRQA QA-C | 9.7 | 5013 | Partially implemented |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Miscellaneous** | STD-010-CPP  [MSC41-C] | Never hard code sensitive information. |

| **Noncompliant Code** |
| --- |
| This example must authenticate to a remote service with a code, using the authenticate() function declared below. It passed the authentication code to this function as a string literal. |
| /\* Returns nonzero if authenticated \*/  **int** authenticate(const **char**\* code);    **int** main() {    if (!authenticate("correct code")) {  **printf**("Authentication error\n");      return -1;    }    **printf**("Authentication successful\n");    // ...Work with system...    return 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution requires the user to supply the authentication code, and securely erases it when done,  Using the memset\_s() function. |
| /\* Returns nonzero if authenticated \*/  **int** authenticate(const **char**\* code);    **int** main() {  #define CODE\_LEN 50  **char** code[CODE\_LEN];  **printf**("Please enter your authentication code:\n");  **fgets**(code, sizeof(code), stdin);  **int** flag = authenticate(code);    memset\_s(code, 0, sizeof(code));    if (!flag) {  **printf**("Access denied\n");      return -1;    }  **printf**("Access granted\n");    // ...Work with system...    return 0;  } |

|  |
| --- |
| **Principles(s):** Hard coding sensitive information, such as passwords or encryption keys can expose the information to attackers. It also increases the need to manage and accommodate changes to the code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | HARDCODED.AUTH  HARDCODED.DNS  HARDCODED.KEY  HARDCODED.SALT | Hardcoded Authentication  Hardcoded DNS Name  Hardcoded Crypto Key  Hardcoded Crypto Salt |
| Parasoft C/C++test | 2020.2 | CERT\_C-MSC41-a | Do not hard code string literals |
| PC-lint Plus | 1.4 | 2460 | Assistance provided: reports when a literal is provided as an argument to a function parameter with the ‘noliteral’ argument Semantic; several Windows API functions are marked as such and the ‘-sem’ option can apply it to other functions as appropriate |
| Polyspace Bug Finder | R2020a | CERT C: Rule MSC41-C | Checks for hard coded sensitive data |

## 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 should take place during the verify and testing phase. This will ensure that the static analysis tools will be implemented during this phase to detect any errors and/or irregularities. By implementing automation early, we can include it into the software being built opposed to redesigning the software around new automations. By including it early, this will ensure these automated tests will also grow with the software, reducing time and costs to implement new methods.

## 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 |
| --- | --- | --- | --- | --- | --- |
| DCL30-C | High | Probable | High | 6 | 2 |
| INT32-C | High | Likely | High | 9 | 2 |
| STR52-CPP | High | Probable | High | 6 | 2 |
| IDS00-J | High | Probable | Medium | 12 | 1 |
| MEM52-CPP | High | Likely | Medium | 18 | 1 |
| ERR06-C | Medium | Unlikely | Medium | 4 | 3 |
| ERR51-CPP | Low | Probable | Medium | 4 | 3 |
| ERR58-CPP | Low | Likely | Low | 9 | 2 |
| FIO06-C | Medium | Probable | High | 4 | 3 |
| MSC41-C | High | Probable | Medium | 12 | 1 |

## Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | This type of encryption includes data that is stored behind a firewall or antivirus software. This basically transforms personal data into another form of data to protect it and add an additional layer of defense. One way to protect access to this data is provide two-factor authentication. |
| Encryption at flight | The process of encrypting data while it is being transmitted. This policy applies because information is extremely vulnerable while it is being transferred. Encryption in this phase should grow with the company as infrastructure increases. |
| Encryption in use | This is when data is accessed by an application for treatment. Two-factor authentication should be used for this type of data encryption. Allow access to the data based on the user’s role. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This validates the users’ credentials and confirms user identity to grant access to the system. This usually consists of a username and password and falls under user logins. |
| Authorization | This is done after successful authentication and determines whether the user is authorized to access the requested resources. This includes the user level of access to the system. This may authorize the addition of new users and the changing of databases. |
| Accounting | This is a form of user entitlement verification which accounts for what the user has access to. This contains the files, subscriptions, networks accessed by the user and their credentials. |

**\***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 | 03/21/2021 | Updated Template | Timothy Kelly |  |
| 1.2 | 04/11/2021 | Final Version | Timothy Kelly |  |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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