**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 | Input from all untrusted data sources must be validated. Following proper data validation techniques can reduce and eliminate a vast majority of vulnerabilities. Be suspicious of all external data sources. |
| 1. Heed Compiler Warnings | Always pay attention to compiler warnings. Compile code using the highest warning level possible. Eliminate warnings and use static and dynamic analysis tools to detect security flaws. |
| 1. Architect and Design for Security Policies | Design software to implement and enforce security policies. If your system requires different privileges at different times, break the software down into smaller subsystems to accommodate the needs of each part. |
| 1. Keep It Simple | To create the most secure software design possible, keep the architecture as simple as possible. The more complex a system gets, the higher the likelihood that there will be bugs and security flaws. |
| 1. Default Deny | To create a more secure piece of software, base access decisions on permission rather than exclusion. Access should be denied by default and the protection scheme will identify conditions under which access is permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least set of privileges necessary to complete the job. Any elevated permissions should only be accessed from the least amount of time required to complete the privileged task. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off – the -shelf components. Attackers may be able to invoke unused functionality through the use of SQL, command, or other injection attacks. |
| 1. Practice Defense in Depth | Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. |
| 1. Use Effective Quality Assurance Techniques | Good QA techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. |
| 1. Adopt a Secure Coding Standard | Develop and apply a secure coding standard for your target development language and platform. |

### 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] | Variables in C++ must be created with care due to the lack of argument checking. Make sure to use a valid format for strings. |

| **Noncompliant Code** |
| --- |
| Mismatches between arguments and conversion specifications may result in undefined behavior. |
| **void** func(**void**) {  **const** **char** \*error\_msg = "Resource not available to user.";  **int** error\_type = 3;    /\* ... \*/  **printf**("Error (type %s): %d\n", error\_type, error\_msg);    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The Compliant solution ensures that the arguments to the printf() function match their conversion specifications |
| **void** func(**void**) {  **const** **char** \*error\_msg = "Resource not available to user.";  **int** error\_type = 3;    /\* ... \*/  **printf**("Error (type %d): %s\n", error\_type, error\_msg);      /\* ... \*/  } |

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

| **Principles(s):** Validate data input |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO47 | Fully implemented |
| CodeSonar | 6.1p0 | IO.INJ.FMT  MISC.FMT  MISC.FMTTYPE | Format string injection  Format String  Format String type error |
| Coverity | 2017.07 | PW | Report when the number of arguments differs from the number of required arguments according to the format string. |
| GCC | 4.3.5 |  | Can detect violations of this recommendation when the -Wformat flag is used |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Ensure you are using the correct data types for integers. |

| **Noncompliant Code** |
| --- |
| The following can result in signed integer overflow during the addition of 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 solution ensures that the addition operation cannot 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;    }    /\* ... \*/  } |

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

| **Principles(s):** Use effective QA standards. Integer overflow can lead to buffer overflows and the execution of arbitrary code by an attacker. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Integer-overflow | Fully Checked |
| CodeSonar | 6.1p0 | **ALLOC.SIZE.ADDOFLOW ALLOC.SIZE.IOFLOW ALLOC.SIZE.MULOFLOW ALLOC.SIZE.SUBUFLOW MISC.MEM.SIZE.ADDOFLOW MISC.MEM.SIZE.BAD MISC.MEM.SIZE.MULOFLOW MISC.MEM.SIZE.SUBUFLOW** | Addition overflow of allocation size Integer overflow of allocation size Multiplication overflow of allocation size Subtraction underflow of allocation size Addition overflow of size Unreasonable size argument Multiplication overflow of size Subtraction underflow of size |
| Coverity | 2017.07 | TAINTED\_SCALAR  BAD\_SHIFT | Implemented |
| Helix QAX | 2021.3 | C2800, C2801, C2802, C2803, C2860, C2861, C2862, C2863  C++2800, C++2801, C++2802, C++2803, C++2860, C++2861, C++2862, C++2863 | Fully Checked |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Guarantee that storage for strings has sufficient space for characters data and the null terminator. |

| **Noncompliant Code** |
| --- |
| Below we can see an off-by-one error. The loop copies data from src to dest but the loop does not account for the null-termination character. |
| #include <stddef.h>    **void** copy(**size\_t** n, **char** src[n], **char** dest[n]) {  **size\_t** i;    **for** (i = 0; src[i] && (i < n); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| The loop termination condition is modified to account for the null-termination character that is appended to dest |
| #include <stddef.h>    **void** copy(**size\_t** n, **char** src[n], **char** dest[n]) {  **size\_t** i;    **for** (i = 0; src[i] && (i < n - 1); ++i) {       dest[i] = src[i];     }     dest[i] = '\0';  } |

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

| **Principles(s):** Keep it simple.  Validate Input Data. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 |  | Supported  Reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR31 | Detects calls to unsafe string functions that may cause buffer overflow. |
| CodeSonar | 6.1p0 | **LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\*** | Buffer overrun Type overrun No space for null terminator A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Coverity | 2017.07 | **STRING\_OVERFLOW**  **BUFFER\_SIZE**  **OVERRUN**  **STRING\_SIZE** | Fully Implemented. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Protect yourself from malicious attacks by making sure sql injection can not happen. |

| **Noncompliant Code** |
| --- |
| The java.sql.PreparedStatement class properly escapes input strings, preventing SQL injection when used correctly. |
| **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** |
| --- |
| 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      }    }  } |

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

| **Principles(s):** Validate Input Data. Use Effective Quality Assurance Techniques. |
| --- |

**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 |
| CodeSonar | 6.1p0 | JAVA.IO.INJ.SQL | SQL Injection |
| 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 |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Only Free memory allocated dynamically |

| **Noncompliant Code** |
| --- |
| This noncompliant code example sets c\_str to reference either dynamically allocated memory or a statically allocated string literal depending on the value of argc. In either case, c\_str is passed as an argument to free(). If anything other than dynamically allocated memory is referenced by c\_str, the call to free(c\_str) is erroneous. |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>    **enum** { MAX\_ALLOCATION = 1000 };    **int** main(**int** argc, **const** **char** \*argv[]) {  **char** \*c\_str = NULL;  **size\_t** len;    **if** (argc == 2) {      len = **strlen**(argv[1]) + 1;  **if** (len > MAX\_ALLOCATION) {        /\* Handle error \*/      }      c\_str = (**char** \*)**malloc**(len);  **if** (c\_str == NULL) {        /\* Handle error \*/      }  **strcpy**(c\_str, argv[1]);    } **else** {      c\_str = "usage: $>a.exe [string]";  **printf**("%s\n", c\_str);    }  **free**(c\_str);  **return** 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution eliminates the possibility of c\_str referencing memory that is not allocated dynamically when passed to free(): |
| #include <stdlib.h>  #include <string.h>  #include <stdio.h>    **enum** { MAX\_ALLOCATION = 1000 };    **int** main(**int** argc, **const** **char** \*argv[]) {  **char** \*c\_str = NULL;  **size\_t** len;    **if** (argc == 2) {      len = **strlen**(argv[1]) + 1;  **if** (len > MAX\_ALLOCATION) {        /\* Handle error \*/      }      c\_str = (**char** \*)**malloc**(len);  **if** (c\_str == NULL) {        /\* Handle error \*/      }  **strcpy**(c\_str, argv[1]);    } **else** {  **printf**("%s\n", "usage: $>a.exe [string]");  **return** EXIT\_FAILURE;    }  **free**(c\_str);  **return** 0;  } |

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

| **Principles(s):** Heed compiler warnings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Invalid-free | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM34 | Can detect memory deallocations for stack objects. |
| Clang | 3.9 | Clang-analyzer-unix.Malloc | Checked by clang-tidy; can detect some instances of this rule, but does not detect all. |
| CodeSonar | 6.1.p0 | ALLOC.FNH | Free non-heap variables |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Utilize assertions to thoroughly test your program |

| **Noncompliant Code** |
| --- |
| This noncompliant code example uses the assert() macro to verify that memory allocation succeeded. Because memory availability depends on the overall state of the system and can become exhausted at any point during a process lifetime, a robust program must be prepared to gracefully handle and recover from its exhaustion. Consequently, using the assert() macro to verify that a memory allocation succeeded would be inappropriate because doing so might lead to an abrupt termination of the process, opening the possibility of a denial-of-service attack. |
| |  | | --- | | **char** \*dupstring(**const** **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char** \*)**malloc**(len + 1);  **assert**(NULL != dup);    **memcpy**(dup, c\_str, len + 1);  **return** dup;  } | |

| **Compliant Code** |
| --- |
| This compliant solution demonstrates how to detect and handle possible memory exhaustion: |
| **char** \*dupstring(**const** **char** \*c\_str) {  **size\_t** len;  **char** \*dup;      len = **strlen**(c\_str);    dup = (**char**\*)**malloc**(len + 1);    /\* Detect and handle memory allocation error \*/  **if** (NULL == dup) {  **return** NULL;    }    **memcpy**(dup, c\_str, len + 1);  **return** dup;  } |

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

| **Principles(s):** Use Effective Quality Assurance Techniques. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Coverity | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect. |
| Parasoft C/C++ test | 2021.2 | CERT\_C-MSC11-a | Assert liberally to document internal assumptions and invariants |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Honor exception specifications |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, a function is declared as nonthrowing, but it is possible for std::vector::resize() to throw an exception when the requested memory cannot be allocated. |
| |  | | --- | | #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) noexcept(**true**) {    v.resize(s); // May throw  } | |

| **Compliant Code** |
| --- |
| In this compliant solution, the function's *noexcept-specification* is removed, signifying that the function allows all exceptions. |
| |  | | --- | | #include <cstddef>  #include <vector>    **void** f(std::vector<**int**> &v, **size\_t** s) {    v.resize(s); // May throw, but that is okay  } | |

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

| **Principles(s):** Heed Compiler Warnings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Unhandled-throw-noexcept | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++ - ERR55 | Partially checked |
| Helix QAX | 2021.2 | C++4035, C++4036, C++4632 | Partially Implemented |
| LDRA tool suite | 9.71 | 56D | Partially implemented |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Error Handling] | [STD-008-CPP] | Do not abruptly terminate the program |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the call to f(), which was registered as an exit handler with std::at\_exit(), may result in a call to std::terminate() because throwing\_func() may throw an exception. |
| |  | | --- | | #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.    throwing\_func();  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } | |

| **Compliant Code** |
| --- |
| In this compliant solution, f() handles all exceptions thrown by throwing\_func() and does not rethrow. |
| |  | | --- | | #include <cstdlib>    **void** throwing\_func() noexcept(**false**);    **void** f() { // Not invoked by the program except as an exit handler.  **try** {      throwing\_func();    } **catch** (...) {      // Handle error    }  }    **int** main() {  **if** (0 != std::**atexit**(f)) {      // Handle error    }    // ...  } | |

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

| **Principles(s):** User Effective Quality Assurance Techniques. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Stdlib-use | Partially checked |
| CodeSonar | 6.1p0 | BADFUNC.ABORT  BADFUNC.EXIT | Use of abort  Use of exit |
| Helix QAC | 2021.2 | C++ 5014 | Partially checked |
| LDRA tool suite | 9.7.1 | 122 S | Enhanced Enforcement |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Integer conversion] | [STD-009-CPP] | Ensure integer conversions do no result in lost or misinterpreted data |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type. |
| |  | | --- | | #include <limits.h>    **void** func(**void**) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;    sc = (**signed** **char**)u\_a; /\* Cast eliminates warning \*/    /\* ... \*/  } | |

| **Compliant Code** |
| --- |
| Validate ranges when converting from an unsigned type to a signed type. This compliant solution can be used to convert a value of unsigned long int type to a value of signed char type: |
| |  | | --- | | #include <limits.h>    **void** func(**void**) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;  **if** (u\_a <= SCHAR\_MAX) {      sc = (**signed** **char**)u\_a;  /\* Cast eliminates warning \*/    } **else** {      /\* Handle error \*/    }  } | |

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

| **Principles(s):** Validate input data. Heed Compiler Warnings. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CppCheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Klocwork | 2021.3 | PORTING.CAST.SIZE | Partially implemented |
| LDRA tool suite | 9.7.1 | 93 S, 433 S, 434 S | Partially Implemented |
| TrustInSoftAnalyzer | 1.38 | Signed\_downcast | Exhaustively verified. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Input Output] | [STD-010-CPP] | Exclude user input from format strings |

| **Noncompliant Code** |
| --- |
| The incorrect\_password() function in this noncompliant code example is called during identification and authentication to display an error message if the specified user is not found or the password is incorrect. The function accepts the name of the user as a string referenced by user. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    **void** incorrect\_password(**const** **char** \*user) {  **int** ret;    /\* User names are restricted to 256 or fewer characters \*/  **static** **const** **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + **sizeof**(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);  **if** (msg == NULL) {      /\* Handle error \*/    }    ret = snprintf(msg, len, msg\_format, user);  **if** (ret < 0) {      /\* Handle error \*/    } **else** **if** (ret >= len) {      /\* Handle truncated output \*/    }  **fprintf**(stderr, msg);  **free**(msg);  } |

| **Compliant Code** |
| --- |
| This compliant solution fixes the problem by replacing the fprintf() call with a call to fputs(), which outputs msg directly to stderr without evaluating its contents: |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    **void** incorrect\_password(**const** **char** \*user) {  **int** ret;    /\* User names are restricted to 256 or fewer characters \*/  **static** **const** **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + **sizeof**(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);  **if** (msg == NULL) {      /\* Handle error \*/    }    ret = snprintf(msg, len, msg\_format, user);  **if** (ret < 0) {      /\* Handle error \*/    } **else** **if** (ret >= len) {      /\* Handle truncated output \*/    }  **fputs**(msg, stderr);  **free**(msg);  } |

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

| **Principles(s):** Validate Input data, Default Deny. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO30 | Partially Implemented |
| CodeSonar | 6.1p0 | IO.INJ.FMT  MISC.FMT | Format String injection  Format string |
| Coverity | 2017.07 | TAINTED\_STRING | Implemented |
| LDRA tool suite | 9.7.1 | 86 D | Partially Implemented |

### 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 best areas in which automation would be helpful would be in testing, health checks, and monitoring. These areas typically require a lot of repetitive code to be written, consuming a lot of development time. With the implementation of automation in these areas, tests could be run and completed at higher speeds than a human would be capable of doing. Development time would increase as well as the quality of software.

### Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-002-CPP | High | Likely | High | P9 | 2 |
| STD-003-CPP | High | Likely | Medium | P18 | 1 |
| STD-004-CPP | High | Probable | Medium | P12 | 1 |
| STD-005-CPP | High | Likely | Medium | P18 | 1 |
| STD-006-CPP | Low | Unlikely | High | P1 | 3 |
| STD-007-CPP | Low | Likely | Low | P9 | 2 |
| STD-008-CPP | Low | Probable | Medium | P4 | 3 |
| STD-009-CPP | High | Probable | High | P6 | 2 |
| STD-010-CPP | High | Likely | Medium | P18 | 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 | Encryption in rest is the process of taking one form of data and transforming it into another form of data to protect sensitive information from falling into the wrong hands. Unauthorized users are unable to decrypt the data without the correct key, keeping it safe. By following this policy and encrypting data not often used, the data will be safe from malicious attackers. |
| Encryption at flight | Encryption-in-Flight is the process of encrypting data while the data is being transmitted. The act of encrypting data in flight will prevent attackers from accessing the information if they should attempt an attack during mid-transfer. This policy should be used in the event that sensitive information is being transferred. |
| Encryption in use | Encryption in use is the act of encrypting data, whether it's in use or at rest. All sensitive data is encrypted throughout its entire life cycle, preventing data loss from occurring. Using this policy will ensure a comprehensive practical approach to protecting all company data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is a method for identifying a user with a user name or password. Typically, authentication can be seen as the first line of defense against unwanted system users. |
| Authorization | Authorization is a means to check the user's permissions within a system. Depending on the user, each may have a different authorization level to a system granting them certain privileges that other users might not have. Authorization can be seen as another useful layer of security which is why it is essential to set permissions accordingly. |
| Accounting | Accounting is a means of monitoring and logging what any particular user is doing within a system. In most cases, accounting can be used to determine what type of authorization a user has within a system and can also be used to spot malicious activity. |

**\***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 | 11/06/2021 | Partially completed template | Anthony Lewandowski | [Insert text.] |
| 1.2 | 12/05/2021 | Completed template | Anthony Lewandowski | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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