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



# Green Pace Secure Development Policy

# By Alex Frankel

## 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 | The idea is that you should never trust input data from any source, even sources you have absolute faith in. Validating input data makes sure that it follows the desired format for example a form that takes an email address or a phone number should validate that the address is in a valid address format ([x@y.com](mailto:x@y.com)). A phone number should only include 10 numerical digits. The common example is an SQL injection attack that attempts to use escape characters to rewrite a query. If the query parameterizes user input, the injected clause will be passed as part of the String. A best practice here would be to attempt to detect the most common forms of injection within the query. |
| 1. Heed Compiler Warnings | Compiler warnings provide the developer with an insight into bugs or undocumented behavior occurring in the code. Early detection of shortcomings in code is valuable as it saves time and anguish down the road. |
| 1. Architect and Design for Security Policies | Design in Depth deals with multilayered systems that have many points of interest that need to be secured. This includes administrative, social policies as well as digital solutions. Social engineering (phishing, scams) can harm just as much as a software vulnerability. Having an active threat model and addressing every individual weakness and *documenting the whole process* is the best strategy today. This task is nontrivial which is why it warrants an architecture surrounding it on the highest level! |
| 1. Keep It Simple | Complicated code isn’t just difficult to understand but also much easier to mess up! The famous KISS method of engineering KEEP IT SIMPLE STUPID! The more straightforward the solution, the easier it is to maintain. |
| 1. Default Deny | By default a system should deny traffic. Rather than allowing all traffic up front, it is much safer to limit access and grant permissions individually and deliberately. |
| 1. Adhere to the Principle of Least Privilege | A member of a digital system should have only the bare minimal administrative privileges warranted them. New employees don’t need admin access to everything. This corresponds to the object oriented coding principle of encapsulation by providing only an interface of functions rather than all of the mechanisms behind them. That interface may be your point person with high privileges so that these requests are handled properly |
| 1. Sanitize Data Sent to Other Systems | Sanitizing data is a process of formatting or vetting input from an outside source. This returns to principle 1: validate user input, where we can see that data provided isn’t always trustworthy. If you are passing data down the line to another system it is your responsibility to make sure it isn’t malicious in any way, and if it isn’t, to provide it in a standard format that the next system can use. Like I said before:  Validate user input follows the correct format.  Using regular expressions on Strings to accomplish this.  Javascript Object Notation is a great example of formatting because it interfaces well with databases like MongoDB. Data that has been formatted by JSON is universally usable for any Javascript implementation. RESTful API can decide whether to accept or reject requests to a web service. |
| 1. Practice Defense in Depth | DID combines all of these principles into a best practice: a multilayered approach for securing any system. These layers work best by overlapping with each other to prevent any one point of contention from compromising the entire system. |
| 1. Use Effective Quality Assurance Techniques | Assuring the quality of code is being held to a high standard is important in securing a system. Extensive testing should be applied including in-depth white box methods as well as integration with larger systems. For this strategy, no stone will go unturned. |
| 1. Adopt a Secure Coding Standard | Developers should be aware of any and all known vulnerabilities for code bases they are using. The biggest example of this is the CERT C and C++ coding standard. CERT addresses vulnerabilities in the underlying architecture of the C and C++ languages. One such vulnerability is Integer wraparound, overflow, and truncation errors. NIST OWASP is a database of known vulnerabilities including and beyond the scope of these languages. |

### 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 | “Integer conversions, both implicit and explicit (using a cast), must be guaranteed not to result in lost or misinterpreted data. This rule is particularly true for integer values that originate from untrusted sources and are used in any of the following ways” INT31-C  Examples of this are truncation issues when converting to a lower width data type or loss of sign when converting a signed value to an unsigned one. |

| **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. This noncompliant code example results in a truncation error on most implementations” |
| #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” |
| 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):** 1: Validate Input Data- input data integrity must be upheld by using appropriate data typing to prevent truncation or overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| CodeSonar | 8.3p0 | LANG.CAST.PC.AV  LANG.CAST.PC.CONST2PTR  LANG.CAST.PC.INT  LANG.CAST.COERCE  LANG.CAST.VALUE  ALLOC.SIZE.TRUNC  MISC.MEM.SIZE.TRUNC  LANG.MEM.TBA | Cast: arithmetic type/void pointer  Conversion: integer constant to pointer  Conversion: pointer/integer  Coercion alters value  Cast alters value  Truncation of allocation size  Truncation of size  Tainted buffer access |
| Cppcheck | 2.15 | memsetValueOutOfRange |  |
| Parasoft C/C++test | 2024.2 | CERT\_C-INT31-a  CERT\_C-INT31-b  CERT\_C-INT31-c  CERT\_C-INT31-d  CERT\_C-INT31-e  CERT\_C-INT31-f  CERT\_C-INT31-g  CERT\_C-INT31-h  CERT\_C-INT31-i  CERT\_C-INT31-j  CERT\_C-INT31-k  CERT\_C-INT31-l  CERT\_C-INT31-m  CERT\_C-INT31-n  CERT\_C-INT31-o  CERT\_C-INT31-p | examples primarily focus on not comparing mismatch datatypes  Avoid data loss when converting between integer types  Avoid value change when converting between integer types |

Ort

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Operations on signed integers may cause them to overflow. It is important to detect this when applying any arithmetic operations to a signed integer value.  INT32-C |

| **Noncompliant Code** |
| --- |
| Signed integer overflow may occur through addition. |
| void func(signed int si\_a, signed int si\_b) {  signed int sum = si\_a + si\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The following solution will test whether an overflow has occurred. This can be handled by the developer as they wish (for example, using a data type with a larger width than INT\_MAX so it can be represented without error). |
| #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):** 2: Heed Compiler Warnings- Uninitialized memory without a value can cause undocumented behavior.  9: Use Effective Quality Assurance Techniques- Assuring that integer values don’t overflow is a good practice. Increasing the width of a variable when necessary combats this. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | Integer-overflow | Fully Checked |
| CodeSonar | 8.3p0 | 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 |
| Klocwork | 2024.4 | NUM.OVERFLOW  CWARN.NOEFFECT.OUTOFRANGE  NUM.OVERFLOW.DF |  |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Strings must have enough storage for character data and the null terminator.  “Copying data to a buffer that is not large enough to hold that data results in a buffer overflow.”  STR31-C |

| **Noncompliant Code** |
| --- |
| The loop does not account for the null-termination character and writes 1 byte past the end of 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); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |

| **Compliant Code** |
| --- |
| The loop terminates at the correct value, allowing space 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 - 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):** 1: Validate Input Data- Prevent overruns and overwrites by detecting improper input  7: Sanitize Data Sent to Other Systems- Sanitize malicious queries, validate user input before sending it elsewhere |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 |  | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| CodeSonar | 8.3p0 | 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 |
| Parasoft C/C++test | 2024.2 | CERT\_C-STR31-a  CERT\_C-STR31-b  CERT\_C-STR31-c  CERT\_C-STR31-d  CERT\_C-STR31-e | Avoid accessing arrays out of bounds  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Avoid using unsafe string functions which may cause buffer overflows |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | SQL injection occurs from escaping from a string clause within a query and appending an extra field for said query in order to access sensitive data. The use of parameterized inputs for queries will prevent the attacker from escaping the clause.  STR02-C |

| **Noncompliant Code** |
| --- |
| The following code creates an sql query by directly concatenating user provided values. These values may escape from the string quotes and add statements to the query.  Example from:  https://stackoverflow.com/questions/36815112/c-and-sqlite-how-to-execute-a-query-formed-by-user-input |
| string sql = "INSERT INTO passwords (ID,title,password) VALUES (" + id + ',' + title + ',' + password + ");";  if(sqlite3\_open("pw.db", &db) == SQLITE\_OK)  {  sqlite3\_prepare( db, sql.c\_str(), -1 &st, NULL);  sqlite3\_step( st );  } |

| **Compliant Code** |
| --- |
| Using a prepared statement will format the value inserted into the query as a String |
| string sql = "INSERT INTO passwords (ID,title,password) VALUES (?,?,?)";  if(sqlite3\_open("pw.db", &db) == SQLITE\_OK)  {  sqlite3\_prepare( db, sql.c\_str(), -1 &st, NULL);  sqlite3\_bind\_int(st, 1, ID);  sqlite3\_bind\_text(st, 2, title.c\_str(), title.length(), SQLITE\_TRANSIENT);  sqlite3\_bind\_text(st, 3, password.c\_str(), password.length(), SQLITE\_TRANSIENT);  sqlite3\_step( st );  } |

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

| **Principles(s):** 1: Validate Input Data- sanitize input to prevent injection attacks, this includes parameterizing/preparing statements for any user input  5: Default Deny- Blacklist injection attempts!  8: Practice Defense In Depth/10: Adopt a secure coding standard- don’t give just any user direct access to SQL database queries. Create a proper RESTful request API so that functionality is properly encapsulated |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 |  | Supported by stubbing/taint analysis |
| CodeSonar | 8.3p0 | IO.INJ.COMMAND  IO.INJ.FMT  IO.INJ.LDAP  IO.INJ.LIB  IO.INJ.SQL  IO.UT.LIB  IO.UT.PROC | Command injection  Format string injection  LDAP injection  Library injection  SQL injection  Untrusted Library Load  Untrusted Process Creation |
| Coverity | 6.5 | TAINTED\_STRING | Fully Implemented |
| Parasoft C/C++test | 2024.2 | CERT\_C-STR02-a  CERT\_C-STR02-b  CERT\_C-STR02-c | Protect against command injection  Protect against file name injection  Protect against SQL injection |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Without proper protections in place, memory can overflow which may allow the execution of arbitrary code defined by an attacker. Undefined behave or can occur when memory allocation results in error or when freed memory is accessed.  MEM30-C |

| **Noncompliant Code** |
| --- |
| p->next refers to memory that has been freed |
| #include <stdlib.h>    struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  for (struct node \*p = head; p != NULL; p = p->next) {  free(p);  }  } |

| **Compliant Code** |
| --- |
| p->next is stored in q before p is freed |
| #include <stdlib.h>    **struct** node {  **int** value;  **struct** node \*next;  };    **void** free\_list(**struct** node \*head) {  **struct** node \*q;  **for** (**struct** node \*p = head; p != NULL; p = q) {      q = p->next;  **free**(p);    }  } |

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

| **Principles(s):** 2: Heed Compiler Warnings- Improper data management can be denoted by the compiler about potential leeks/unfreed/unlawful memory usage.  10: Adopt a secure coding standard- Manage memory appropriately when coding |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | Dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| CodeSonar | 8.3p0 | ALLOC.UAF | Use After Free |
| Coverity | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Assertions are a white-box testing strategy that can be utilized throughout one’s code in order to determine correct functionality. For example, they can be used to test the value of any statement’s actual value versus expected values.  ERR06-C Understand the termination behavior of assert() and abort()  Exceptions should be handled before assertion statements |

| **Noncompliant Code** |
| --- |
| “Assertion statements are not a substitute for error-handling code, however. The following example shows an assertion statement that can lead to problems in the final release code:” |
| myErr = myGraphRoutine(a, b);  /\* No Code to handle errors \*/  ASSERT(!myErr); // Don't do this!  \_ASSERT(!myErr); // Don't do this, either! |

| **Compliant Code** |
| --- |
| You can use assertions to test for error conditions at a point in your code where any errors should have been handled. In the following example, a graphic routine returns an error code or zero for success. |
| myErr = myGraphRoutine(a, b);  /\* Code to handle errors and  reset myErr if successful \*/  ASSERT(!myErr); -- MFC version  \_ASSERT(!myErr); -- CRT version |

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

| **Principles(s):** 4: Keep it Simple/9: Use Effective Quality Assurance Techniques- Assertions are meant to test expected values, not to replace handling errors. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | **bad-function bad-macro-use** | Supported |
| Parasoft C/C++test | 2024.2 | CERT\_C-ERR06-a | Do not use assertions  Generally, production code will have assertions or other tests omitted |
| Compass/ROSE |  |  | Can detect some violations of this rule. However, it can only detect violations involving abort() because assert() is implemented as a macro |
| RuleChecker | 24.04 | bad-function  bad-macro-use | Supported |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Unhandled exceptions will terminate a program abruptly!  Exceptions must be handled!  The two following standards  ERR50-CPP. Do not abruptly terminate the program  ERR51-CPP. Handle all exceptions  Go hand-in-hand to create a coding policy that accounts for all possibilities. |

| **Noncompliant Code** |
| --- |
| None of the following functions catch the exception. When it throws, the program is terminated. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| The following code surrounds the function that may throw an exception with the try/catch block. The program will not automatically terminate and the developer may now decide what to do when exceptions are thrown. |
| 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):** 10: Adopt a Secure Coding Standard- Always catch exceptions, this is a coding best practice. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |
| CodeSonar | 8.3p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Parasoft C/C++test | 2024.2 | 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 |
| RuleChecker | 22.10 | **main-function-catch-all early-catch-all** | Partially Checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | STD-008-CPP | “Never call a formatted I/O function with a format string containing a tainted value… An attacker who can fully or partially control the contents of a format string can crash a vulnerable process, view the contents of the stack, view memory content, or write to an arbitrary memory location. Consequently, the attacker can execute arbitrary code with the permissions of the vulnerable process [Seacord 2013b]. Formatted output functions are particularly dangerous because many programmers are unaware of their capabilities. For example, formatted output functions can be used to write an integer value to a specified address using the %n conversion specifier.” FIO30-C |

| **Noncompliant Code** |
| --- |
| The following code passes untrusted user input to fprintf which could cause a format-string vulnerability from the resulting overflow (if user has more than 256 characters) |
| #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** |
| --- |
| Using fputs instead of fprintf will output a string without evaluating it… |
| #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):** 1: Validate User Input/7:Sanitize data sent to other systems- Check for injection or overflows when taking user input. Do not forward this input unless it has been properly sanitized or handled first. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 |  | Supported via stubbing/taint analysis |
| CodeSonar | 8.3p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| Parasoft C/C++test | 2024.2 | CERT\_C-FIO30-a  CERT\_C-FIO30-b  CERT\_C-FIO30-c | Avoid calling functions printf/wprintf with only one argument other than string constant  Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable  Never use unfiltered data from an untrusted user as the format parameter |
| Klocwork | 2024.4 | **SV.FMTSTR.GENERIC SV.TAINTED.FMTSTR** |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Miscellaneous (RANDOM NUMBERS) | STD-090-CPP | When not utilized properly, procedurally generated pseudorandom numbers can become entirely predictable!  “The C Standard rand() function, exposed through the C++ standard library through <cstdlib> as std::rand(), makes no guarantees as to the quality of the random sequence produced.” MSC50-CPP |

| **Noncompliant Code** |
| --- |
| Using std::rand() to generate a random ID is predictable |
| #include <cstdlib>  #include <string>    void f() {  std::string id("ID"); // Holds the ID, starting with the characters "ID" followed  // by a random integer in the range [0-10000].  id += std::to\_string(std::rand() % 10000);  // ...  } |

| **Compliant Code** |
| --- |
| Instead, make the process of selection non-trivial.  “It breaks random number generation into two parts: one is the algorithm responsible for providing random values (the engine), and the other is responsible for distribution of the random values via a density function (the distribution).” |
| #include <random>  #include <string>    **void** f() {    std::string id("ID"); // Holds the ID, starting with the characters "ID" followed                          // by a random integer in the range [0-10000].    std::uniform\_int\_distribution<**int**> distribution(0, 10000);    std::random\_device rd;    std::mt19937 engine(rd());    id += std::to\_string(distribution(engine));    // ...  } |

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

| **Principles(s):** 9: Use Effective Quality Assurance Techniques- Do not use deprecated functions or functions that are otherwised warned against. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | bad-function (AUTOSAR.26.5.1A) | Fully checked |
| CodeSonar | 8.3p0 | BADFUNC.RANDOM.RAND | Use of rand |
| Parasoft C/C++test | 2024.2 | **CERT\_CPP-MSC50-a** | Do not use the rand() function for generating pseudorandom numbers |
| Polyspace Bug Finder | R2024a | CERT C++: MSC50-CPP | Checks for use of vulnerable pseudo-random number generator (rule partially covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Object Oriented Programming | STD-010-CPP | Objects may be compared through bytewise valuations. Doing so with C Standard Library Functions may result in undefined behaviors.  OOP57-CPP |

| **Noncompliant Code** |
| --- |
| Memset is being called to re-initialize the object C. This is not the proper way to re-initialize and may disrupt class invariants. |
| #include <cstring>  #include <iostream>    class C {  int scalingFactor;  int otherData;    public:  C() : scalingFactor(1) {}    void set\_other\_data(int i);  int f(int i) {  return i / scalingFactor;  }  // ...  };    void f() {  C c;    // ... Code that mutates c ...    // Reinitialize c to its default state  std::memset(&c, 0, sizeof(C));    std::cout << c.f(100) << std::endl;  } |

| **Compliant Code** |
| --- |
| The class is re-initialized within a function clear() that initializes a new instance of the desired object and swaps the reference to that new object. |
| #include <iostream>  #include <utility>    class C {  int scalingFactor;  int otherData;    public:  C() : scalingFactor(1) {}    void set\_other\_data(int i);  int f(int i) {  return i / scalingFactor;  }  // ...  };    template <typename T>  T& clear(T &o) {  using std::swap;  T empty;  swap(o, empty);  return o;  }    void f() {  C c;    // ... Code that mutates c ...    // Reinitialize c to its default state  clear(c);    std::cout << c.f(100) << std::endl;  } |

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

| **Principles(s):** 10: Adopt a Secure Coding Standard- Be careful when comparing objects such as in this case. Perhaps compare their fields instead of objects directly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | stdlib-use-ato  stdlib-use  stdlib-use-getenv  stdlib-use-system  include-time  stdlib-use-string-unbounded | Partially Checked |
| CodeSonar | 8.3p0 | BADFUNC.MEMCMP  BADFUNC.MEMSET | Use of memcmp  Use of memset |
| Parasoft C/C++test | 2024.2 | CERT\_CPP-OOP57-a  CERT\_CPP-OOP57-b | Do not initialize objects with a non-trivial class type using C standard library functions  Do not compare objects of nonstandard-layout class type with C standard library functions |
| Polyspace Bug Finder | R2024a | CERT C++: OOP57-CPP | Checks for bytewise operations on nontrivial class object (rule fully covered) |

### 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.



DevSecOps is the inclusion of security into every facet of the development process. Before engaging in a CI/CD development process, requirements engineering is generally the first step in designing a system. The addition of Security Quality Requirements Engineering (SQUARE methodology) into the process ensures that quality standards are being upheld. This involves identifying potential vulnerabilities and designing around them immediately and then continuously.

Some of the processes of SQUARE include:

Identifying assets (sensitive data/systems) and security goals (preventing attacks/vulnerabilities).

And Developing Artifacts: use/misuse cases and documenting the entire process.

This is an iterative design process designed specifically for multifaceted Defense in Depth strategies.

All of the aforementioned security policies have been documented by CERT. Multiple Automation tools have been provided that detect these kinds of vulnerabilities in code. Using one or more of these tools ensures that any rule of the CERT guideline ON TOP OF what has been mentioned in this document will be detected and enforced. Static Analysis would fall under the ‘monitor and detect’ section of the diagram.

### 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 | Probable | High | P6 | L2 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Likely | Medium | P18 | L1 |
| STD-009-CPP | Medium | Unlikely | Low | P6 | L2 |
| STD-010-CPP | High | Probable | High | P6 | 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 at rest | Encryption at rest is, like it says, encryption used to protect data at rest (not in transit, not currently being accessed). It keeps data safe until properly authorized use is required of it. A data breach may not compromise anything if data is encrypted at rest. |
| Encryption in flight | Encryption in ‘flight’ is another word for information in transit between two locations. This usually refers to internet data transfer, not physically. If the data is intercepted, it will be unusable. Ideally, this ensures that only the sender and the recipient will be able to access it. |
| Encryption in use | Also known as ‘runtime encryption’, encryption in use refers to encrypting data that is currently being processed in a system. Someone can run calculations on data without requiring the trust of the client because they don’t need access to the data itself but instead as an input/output of a system. An example is hashing an input password to compare it against the hashed value stored in the system. The system itself does not know the unencrypted password, but it can determine whether it is correct or not in this way. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process or processes involved in verifying the identity of a user of a system, before they are granted access. This person can prove themselves by the use of a password, 2-factor authentication, or biometric data.  Key Idea- User Logins. |
| Authorization | Authorization is the subsequent access granted following successful Authentication. A user may be able to access the systems such as databases or files. An administrative user may have more permission that allow them to create new users or change the underlying structure of a database. The idea is that a given user may not necessarily have full access, therefore they will have to request access via proper channels.  Key Idea- User level of access. |
| Accounting | Accounting is the process of documenting a transactional history of all actions a user performs on a system. In short, it is to be ‘accountable’ for anything a user might do. Users have limited access to prevent security breaches. Accounting helps detect these instances. |

**\***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 | 2/2/25 | Module 3 Milestone | Alex Frankel |  |
| 2.0 | 2/16/25 | Completion! | Alex Frankel |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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