**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 6](#_Toc52464060)

[Coding Standard 3 8](#_Toc52464061)

[Coding Standard 4 9](#_Toc52464062)

[Coding Standard 5 11](#_Toc52464063)

[Coding Standard 6 13](#_Toc52464064)

[Coding Standard 7 15](#_Toc52464065)

[Coding Standard 8 17](#_Toc52464066)

[Coding Standard 9 19](#_Toc52464067)

[Coding Standard 10 21](#_Toc52464068)

[Defense-in-Depth Illustration 23](#_Toc52464069)

[Project One 23](#_Toc52464070)

[1. Revise the C/C++ Standards 23](#_Toc52464071)

[2. Risk Assessment 23](#_Toc52464072)

[3. Automated Detection 23](#_Toc52464073)

[4. Automation 23](#_Toc52464074)

[5. Summary of Risk Assessments 24](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 24](#_Toc52464076)

[7. Map the Principles 26](#_Toc52464077)

[Audit Controls and Management 27](#_Toc52464078)

[Enforcement 27](#_Toc52464079)

[Exceptions Process 27](#_Toc52464080)

[Distribution 28](#_Toc52464081)

[Policy Change Control 28](#_Toc52464082)

[Policy Version History 28](#_Toc52464083)

[Appendix A Lookups 28](#_Toc52464084)

[Approved C/C++ Language Acronyms 28](#_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 all input from data sources, especially untrusted ones. Input validation can remove many software vulnerabilities. |
| 1. Heed Compiler Warnings | Use the highest warning level for the compiler being used and change the code to remove warnings. |
| 1. Architect and Design for Security Policies | Design software able to implement and enforce current and future security policies. |
| 1. Keep It Simple | Design as simple as possible to reduce the chance of errors in a more complicated system. |
| 1. Default Deny | Deny access by default and determine what is permitted instead of the other way around. |
| 1. Adhere to the Principle of Least Privilege | Give the least set of privileges necessary to complete the task and any permission should be for the least amount of time required. |
| 1. Sanitize Data Sent to Other Systems | Any data sent to complex systems should be sanitized to prevent against SQL and other injection attacks. |
| 1. Practice Defense in Depth | Use multiple defense strategies to provide extra protection if one layer is broken through. |
| 1. Use Effective Quality Assurance Techniques | Techniques like pen testing will help to identify vulnerabilities before they get introduced into a larger system. |
| 1. Adopt a Secure Coding Standard | Follow a set of guidelines and best practices to promote secure software development. |

### 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** | DCL12-C | Implement abstract data types using opaque types |

| **Noncompliant Code** |
| --- |
| The string\_mx type is visible to the user of the data type from including the string\_m.h file. This can lead to manipulating the fields within the structure. |
| struct string\_mx {  size\_t size;  size\_t maxsize;  unsigned char strtype;  char \*cstr;  };    typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| String\_mx is defined in the header file but not visible to a user of the data abstraction. This keeps the data type private. |
| struct string\_mx;  typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/  //in header file  struct string\_mx {  size\_t size;  size\_t maxsize;  unsigned char strtype;  char \*cstr;  }; |

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

| **Principles(s):**  4. Keep it simple – opaque types simplify interface and hides internal implementation details which makes the design easier and less error prone  10. Adopt a secure coding standard – knowing to keep data types private will improve overall code security |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 104 D | Partially implemented |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | CERT C: Rec. DCL12-C | Checks for structure or union object implementation visible in file where pointer to this object is not dereferenced (rule partially covered) |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | CERT\_C-DCL12-a | If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT31-C | Ensure that integer conversions do not result in lost or misinterpreted data. |

| **Noncompliant Code** |
| --- |
| Type range errors like loss of data or sign can happen when converting an unsigned integer to a signed integer. Can lead to truncation. |
| #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 to signed data value to reduce errors. |
| #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):** 1. Validate input data – reduces the risk of data manipulation and prevents vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **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 |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity)\* | 2017.07 | **NEGATIVE\_RETURNS**  **REVERSE\_NEGATIVE**  **MISRA\_CAST** | Can find array accesses, loop bounds, and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior  Can find instances where a negativity check occurs after the negative value has been used for something else  Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **memsetValueOutOfRange** | The second argument to memset() cannot be represented as unsigned char |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR10-C | Do not concatenate different type of string literals. |

| **Noncompliant Code** |
| --- |
| The code concatenates wide and narrow string literals, which leads to undefined behavior. |
| wchar\_t \*msg = L"This message is very long, so I want to divide it "  "into two parts."; |

| **Compliant Code** |
| --- |
| All elements for a wide string literal concatenation must be a wide string literal. |
| wchar\_t \*msg = L"This message is very long, so I want to divide it "  L"into two parts."; |

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

| **Principles(s):**  4. Keep it simple – reduces the risk of unintended behaviors from incompatible data types  9 Use effective quality assurance techniques – Helps to identify and fix vulnerabilities before they lead to security issues |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | **encoding-mismatch** | Fully checked |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **450 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-STR10-a** | Narrow and wide string literals shall not be concatenated |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **707** | Fully supported |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STR02-C | Sanitize data passed to complex subsystems. |

| **Noncompliant Code** |
| --- |
| Data sanitization requires understanding of the data being passed and the capabilities of the subsystem. User could inject code to retrieve email and passwords. |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| Ensure that all data is valid and accepted. Reject any potentially dangerous data. Defining certain characters that are allowed to be used in the subsystem will help to validate entered data. |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz"  "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  "1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp = user\_data; /\* Cursor into string \*/  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, ok\_chars); cp != end; cp += strspn(cp, ok\_chars)) {  \*cp = '\_';  } |

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

| **Principles(s):** 7. Sanitize data sent to other systems – data must be sanitized to protect against SQL attacks and sanitizing the data will prevent most SQL attacks that lead to wider system breaches |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | 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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | 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 |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | CERT C: Rec. STR02-C | Checks for:   * Execution of externally controlled command * Command executed from externally controlled path * Library loaded from externally controlled path   Rec. partially covered. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM30-C | Do not access freed memory. |

| **Noncompliant Code** |
| --- |
| P is freed before p-> next is executed so it reads memory that has already 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 freeing p to make sure it accesses correct memory. |
| #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):**  9. Use effective quality assurance techniques – coding standards like code static code analysis should help solve these issues  10. Adopt a secure coding standard – contributes to overall more secure practices by mitigating memory related vulnerabilities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 23.04 | Dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | ALLOC.UAF | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/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** | DCL03-C | Use a static assertion to test the value of a constant expression. |

| **Noncompliant Code** |
| --- |
| The assertion needs to be placed in a function and executed. The assertion occurs only at runtime and only if the code path with the assertion is reached. |
| #include <assert.h>    struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| A preprocessor conditional statement can be used when involving only one constant expression. #error gives clear diagnostic messages and evaluates assertions at runtime, which gives no runtime penalty. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

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

| **Principles(s):**  1. Validate input data – static assertions ensure the expression being tested adheres to expected values  10. Adopt a secure coding standard – ensures the code meets predefined security guidelines and best practices |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **(customization)** | Users can implement a custom check that reports uses of the assert() macro |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Could detect violations of this rule merely by looking for calls to assert(), and if it can evaluate the assertion (due to all values being known at compile time), then the code should use static-assert instead; this assumes ROSE can recognize macro invocation |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL03** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | Handle all exceptions. |

| **Noncompliant Code** |
| --- |
| f() and main() both do not catch exceptions thrown by throwing\_func(). Code is terminated as a result of no matching handler. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| main() handles all exceptions to ensure resources are used properly. |
| 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):**  2. Heed compiler warnings – a compiler would most likely show an error on this which should be listened to  9. Use effective quality assurance techniques - handling all exceptions ensures that code follows a consistent and secure approach to exception management |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **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](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2023a | [CERT C++: ERR51-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr51cpp.html) | Checks for unhandled exceptions (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 22.10 | **main-function-catch-all early-catch-all** | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input/Output | FIO46-C | Do not access a closed file. |

| **Noncompliant Code** |
| --- |
| The stdout stream is used after it is closed. This can lead to undefined behavior. |
| #include <stdio.h>    int close\_stdout(void) {  if (fclose(stdout) == EOF) {  return -1;  }    printf("stdout successfully closed.\n");  return 0;  } |

| **Compliant Code** |
| --- |
| Stdout is not used again after it is closed. Stdout can only be assigned the address of an open file. |
| #include <stdio.h>    int close\_stdout(void) {  if (fclose(stdout) == EOF) {  return -1;  }    fputs("stdout successfully closed.", stderr);  return 0;  } |

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

| **Principles(s):**  1. Validate input data – checking the state before attempting to access the file will prevent crashes and other bugs  8. Practice defense in depth – adding multiple layers of security before accessing a file is good practice to prevent errors and potential malicious activities |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **IO.UAC** | Use after close |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-FIO46-a** | Do not use resources that have been freed |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **2471** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule FIO46-C](https://www.mathworks.com/help/bugfinder/ref/certcrulefio46c.html) | Checks for use of previously closed resource (rule partially covered) |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Handling | ERR33-C | Detect and handle standard library errors. |

| **Noncompliant Code** |
| --- |
| The utf8\_to\_wcs() function tries to convert UTF-8 characters to wide characters by using setlocale() but does not check for failure. The setlocale() function will fail if it returns a null pointer among other problems. |
| #include <locale.h>  #include <stdlib.h>    int utf8\_to\_wcs(wchar\_t \*wcs, size\_t n, const char \*utf8,  size\_t \*size) {  if (NULL == size) {  return -1;  }  setlocale(LC\_CTYPE, "en\_US.UTF-8");  \*size = mbstowcs(wcs, utf8, n);  return 0;  } |

| **Compliant Code** |
| --- |
| The setlocale() value is checked to make sure it is valid. It can also return locale to the initial settings. |
| #include <locale.h>  #include <stdlib.h>    int utf8\_to\_wcs(wchar\_t \*wcs, size\_t n, const char \*utf8,  size\_t \*size) {  if (NULL == size) {  return -1;  }  const char \*save = setlocale(LC\_CTYPE, "en\_US.UTF-8");  if (NULL == save) {  return -1;  }    \*size = mbstowcs(wcs, utf8, n);  if (NULL == setlocale(LC\_CTYPE, save)) {  return -1;  }  return 0;  } |

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

| **Principles(s):**  9. Use effective quality assurance techniques – static analysis should address potential errors  10. Adopt a secure coding standard – reduces the risk of errors in code which increases application security |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **LANG.FUNCS.IRV LANG.ERRCODE.NOTEST LANG.ERRCODE.NZ** | Ignored return value Missing Test of Error Code Non-zero Error Code |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-ERR33-a** **CERT\_C-ERR33-b** **CERT\_C-ERR33-d** | The value returned by a standard library function that may return an error should be used The standard library functions for which return values need not be checked should be cast to 'void' Always check the returned value of non-void function |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule ERR33-C](https://www.mathworks.com/help/bugfinder/ref/certcruleerr33c.html) | Checks for:   * Errno not checked * Return value of a sensitive function not checked * Unprotected dynamic memory allocation   Rule partially covered. |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **pointer arithmetic** | Exhaustively verified |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Sensitive Information | MSC41-C | Never hard code sensitive information. |

| **Noncompliant Code** |
| --- |
| The code authenticates to a remote service with a code by using authenticate(). The code is passed as a string literal, which can easily be found through the binary. |
| /\* 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** |
| --- |
| The user must supply the authentication code and it is erased securely when done by using memset\_s(). |
| /\* 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, sizeof(code), 0, sizeof(code));  if (!flag) {  printf("Access denied\n");  return -1;  }  printf("Access granted\n");  // ...Work with system...  return 0;  } |

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

| **Principles(s):**  3. Architect and design for security principles – Making sure sensitive data is not hard coded will help with current and future security policies  5. Default deny – deny who can see sensitive data by default  6. Adhere to the principle of least privilege – only give new users the least amount of privileges  10. Adopt a secure coding standard – reduces the risk of data leaks and other attacks |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 7.4p0 | **HARDCODED.AUTH HARDCODED.DNS HARDCODED.KEY HARDCODED.SALT HARDCODED.SEED** | Hardcoded Authentication Hardcoded DNS Name Hardcoded Crypto Key Hardcoded Crypto Salt Hardcoded Seed in PRNG |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MSC41-a** | Do not hard code string literals |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/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](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2023a | [CERT C: Rule MSC41-C](https://www.mathworks.com/help/bugfinder/ref/certcrulemsc41c.html) | Checks for hard coded sensitive data (rule partially 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.



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 automation used to enforce the standards should be incorporated into the DevSecOps process in between “Design” and “Build”. This will make it, so the automation has already been made a part of the design. It will be discussed within the “assess and plan” phase which will help determine what tests need to be added. The “design” phase will start some of the test-driven design and will then be built securely in the “build” phase. Either incorporating it into these two phases or making its own in between will help the tests be in place when it is later tested and verified. This should give the team the time to implement the automated tests properly and give them enough time to respond to any issues that those tests bring up.

### 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 |
| --- | --- | --- | --- | --- | --- |
| DCL03-C | Low | Unlikely | High | P1 | L3 |
| DCL12-C | Low | Unlikely | High | P1 | L3 |
| INT31-C | Likely | High | High | P9 | L2 |
| STR02-C | High | Likely | Medium | P18 | L1 |
| STR10-C | Low | Probable | Medium | P4 | L3 |
| MEM30-C | High | Likely | Medium | P18 | L1 |
| FIO46-C | Medium | Unlikely | Medium | P4 | L3 |
| ERR33-C | High | Likely | Medium | P18 | L1 |
| ERR51-CPP | Low | Probable | Medium | P4 | L3 |
| MSC41-C | High | Probable | Medium | P12 | L1 |
|  |  |  |  |  |  |

### 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 a security measure used to protect data that is stored or at rest on a device or server. This encryption process converts plain text data into a cipher text, making it unreadable without the decryption key. It is crucial for safeguarding sensitive information from unauthorized access or data breaches. This policy is typically applied to databases, files, and other stored data to ensure data confidentiality when it is not actively being accessed or used. In case an attacker gains unauthorized access to the storage medium, the encrypted data will remain secure and unreadable. |
| Encryption at flight | Encryption in flight is used to secure data as it travels over networks or other communication channels. When data is transmitted between systems or devices, it is vulnerable to interception and eavesdropping. Encryption in flight employs cryptographic algorithms to encode the data during transmission, ensuring that only authorized parties with the decryption key can decipher the information. This applies whenever data is sent over the internet, local networks, or any other communication medium, safeguarding it from potential attackers who might try to steal and exploit the information during transit. |
| Encryption in use | Encryption in use allows data to remain encrypted even while it is being processed or used by applications or algorithms. This encryption technique ensures that sensitive data is protected at all stages, including when it is in use by software or computing systems. It enables secure data processing in scenarios where privacy and confidentiality are important, such as in cloud computing environments or when outsourcing data processing tasks to third-party services. While this encryption method is powerful, it is computationally intensive and often slower than traditional encryption, so it is usually employed in highly sensitive applications with specific privacy requirements. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication focuses on verifying the identity of users, devices, or entities attempting to access a system or resource. It involves validating credentials such as usernames, passwords, or biometrics to ensure that only authorized individuals gain access. This strategy is used during the login process to establish trust and prevent unauthorized access to sensitive information or resources. The policy applies whenever there is a need to control and track user access, protecting against potential security breaches and unauthorized use. |
| Authorization | Authorization involves granting appropriate access rights and permissions to authenticated users or entities. It ensures that users only have access to the resources and actions they are entitled to, based on their role, job function, or privileges. This is used to enforce security policies and control user actions within the system. The applies throughout the user's session to maintain data confidentiality and prevent unauthorized activities or data manipulation. |
| Accounting | Accounting involves monitoring and logging user activities and system events to create a record of actions taken within the system. This helps to trace actions back to specific users and detect potential security incidents or policy violations. Accounting is used for compliance, forensics, and incident response purposes, providing an essential trail of evidence to investigate security breaches or any unauthorized activities within the system. The policy applies continuously to maintain a comprehensive and reliable audit trail, ensuring accountability and transparency in the system's operations. |

**\***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 |  |
| 2.0 | 07/14/2023 | Added 10 security principles and 10 coding standards tables | Tyler Primas |  |
| 3.0 | 08/04/2023 | Finalized document by completing all remaining sections | Tyler Primas |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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