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



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_heading=h.or1rhs47034v)

[Purpose 2](#_heading=h.40k1o1kolhls)

[Scope 2](#_heading=h.nzdlhoajgkp6)

[Module Three Milestone 2](#_heading=h.2dgetfwznn82)

[Ten Core Security Principles 2](#_heading=h.h1jgdvqrmqs)

[C/C++ Ten Coding Standards 3](#_heading=h.hqka23ty8yqj)

[Coding Standard 1 4](#_heading=h.49wdahh9pq1)

[Coding Standard 2 5](#_heading=h.3n9npzu1uunt)

[Coding Standard 3 6](#_heading=h.xeebfhn100im)

[Coding Standard 4 7](#_heading=h.kax43ojdmtcl)

[Coding Standard 5 8](#_heading=h.abrh35dd3sty)

[Coding Standard 6 9](#_heading=h.gvrlqmdyy1si)

[Coding Standard 7 10](#_heading=h.2494rv8g1qi3)

[Coding Standard 8 11](#_heading=h.420ks77ig4gd)

[Coding Standard 9 13](#_heading=h.4lo9qhyr5b37)

[Coding Standard 10 14](#_heading=h.ltigy81cq1pk)

[Defense-in-Depth Illustration 15](#_heading=h.lw0cmznzacrh)

[Project One 15](#_heading=h.1skzrg8u5b9t)

[1.](#_heading=h.pi4z247kore) Revise the C/C++ Standards 15

[2.](#_heading=h.37dgvs9qmeq6) Risk Assessment 15

[3.](#_heading=h.4so9m6ublnj9) Automated Detection 15

[4.](#_heading=h.i3gzykd5ahfi) Automation 15

[5.](#_heading=h.zalwhmm0m3ke) Summary of Risk Assessments 16

[6.](#_heading=h.xf5o2kz141k4) Create Policies for Encryption and Triple A 16

[7.](#_heading=h.2vfgv1gwbf17) Map the Principles 17

[Audit Controls and Management 18](#_heading=h.dpjgw751f0sp)

[Enforcement 18](#_heading=h.o8ltjqifh67h)

[Exceptions Process 18](#_heading=h.cay43vin0jvf)

[Distribution 19](#_heading=h.i1ptfcoebd80)

[Policy Change Control 19](#_heading=h.wh0f4qgfu9nu)

[Policy Version History 19](#_heading=h.nh0k8li4thu9)

[Appendix A Lookups 19](#_heading=h.s6ye44a8blyy)

[Approved C/C++ Language Acronyms 19](#_heading=h.m0stc3cozeb1)

## 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 | User input should be validated before being used since it can be a security risk. Validation can prevent SQL injection or buffer overflows since it checks that the input is formatted correctly. This principle protects against unexpected behaviors by enforcing the necessary constraints the system needs. |
| 1. Heed Compiler Warnings | Heed compiler warnings show unstable, unsafe, or anything else that would be a threat. The compiler should be configured to display very strict warnings and they should be taken care of during development to avoid security issues at the start. If the warnings are ignored, bugs can happen and go unnoticed leading to vulnerabilities. |
| 1. Architect and Design for Security Policies | Security needs to be implemented at the start of the project so that input can be authenticated and can be encrypted. If it’s secure at the start, the system will be better so it should use various methods like authentication, authorization and regular auditing. |
| 1. Keep It Simple | Keeping the design and code simple will prevent bugs from happening in the system. Sometimes more complex solutions are more difficult to test and make it less secure. Simple code lowers the risk of bugs and insecure logic, it also makes it easier for developers working on the code to build on it. |
| 1. Default Deny | Access to any resources should be denied by default, and only given in very specific situations and roles. This will limit the risk of access given on accident because of a coding mistake. |
| 1. Adhere to the Principle of Least Privilege | Each element should have the most limited amount of access possible in order to operate effectively in order to reduce any potential exploitation. This will reduce the impact of a security breach if it happens and prevent escalation. |
| 1. Sanitize Data Sent to Other Systems | Any data shared with third-party systems, databases, or APIs should be checked for malicious content to prevent SQL injection or other attacks. This will prevent payload injections or bad data. |
| 1. Practice Defense in Depth | There should be multiple layers of security to best protect this system. If one layer fails, the other layers should provide protection. Layered protection can offer security against known or unknown threats. This could include protections like input validation, authentication and error handling. |
| 1. Use Effective Quality Assurance Techniques | Complete testing, like unit or integration tests should be completed early to detect bugs or missed cases. A thorough testing process will improve the security of the system and will ensure that insecure code is not used in production. |
| 1. Adopt a Secure Coding Standard | The code should follow secure coding standards like the SEI CERT C++ guidelines so that it follows consistent practices for all teams. This is the simplest way to create safe and maintainable code. |

### 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] | INT35-C. Use correct integer precisions  Using the correct integer precision is important so that variables can represent the values they need to store. If you choose an integer type that is too small, it can cause overflow which causes security vulnerabilities. However, using a type that is too large can slow down the program. Using this standard improves predictability and avoids bugs that may arise.  https://wiki.sei.cmu.edu/confluence/display/c/INT35-C.+Use+correct+integer+precisions |

| **Noncompliant Code** |
| --- |
| The code uses int to store a value that may be too large causing overflow. |
| Int file = getFile();  If (file > 2000000000) {  } |

| **Compliant Code** |
| --- |
| The code uses int\_64 so that the file is not too large and does not cause an overflow. |
| Int64\_t file = getFile();  If (file > 2000000000) {  } |

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

| **Principles(s):** 1. Validate Input Data  Validating input checks that data is the right integer type, prevents overflow and that the right number of decimal places are used.  9. Use Effective Quality Assurance Techniques  Quality assurance techniques verifies that integer types are sized right and prevent errors from being moved into production. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High - overflows can lead to the code failing. | Likely - this is a common mistake. | Low - switching out the data types is pretty easy to do during the coding process | High - it is easy to fix and will prevent bugs. | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.16.0 | Int\_overflow, type\_mismatch | This tool can check for integer overflow. |

Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | INT31-C. Ensure that integer conversions do not result in lost or misinterpreted data  Poor integer conversions can cause lost or misinterpreted values, like turning a large positive number into a negative one. This can cause bugs and as a result poses a security risk. This rule checks that data stays accurate and behaves as needed.  https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data |

| **Noncompliant Code** |
| --- |
| This code converts an unsigned int to a signed int, if the value is too high the result will lead to errors |
| Void process(unsigned int input) {  Int length = input;    If (length < 0) {  Cout << “Length is negative.” << endl;  }  char\* buffer = new char[length];  } |

| **Compliant Code** |
| --- |
| The code checks for an unsigned int so that it can safely fit in a signed int before converting it |
| void process(unsigned int input) {  if (input > static\_cast<unsigned int> (std::numeric\_limits<int>::max())) {  std::cerr << "Input too large to safely convert to int" << endl;  return;  }  int length = static\_cast<int>(input);  char\* buffer = new char[length]; |

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

| **Principles(s):** 1. Validate Input Data  Input validation checks that values are safe to convert, stopping a negative value from happening.  10.Adopt a Secure Coding Standard  This ensures safe practices are used to convert between integer types, which prevents data loss or bugs. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High - data loss can cause a crash of the code. | Likely - this happens a lot in C++ development. | Low - easy to limit the threat with proper boundary checking in the conversions. | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.16.0 | Integer\_conversion, truncation\_warning | Cppcheck can show potential truncation and mismatched data when variables are assigned. |

Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | STR30-C. Do not attempt to modify string literals  String literals are usually stored in read-only memory and trying to change them can cause it to crash. This standard will prevent errors and check that modified strings are changed correctly.  https://wiki.sei.cmu.edu/confluence/display/c/STR30-C.+Do+not+attempt+to+modify+string+literals |

| **Noncompliant Code** |
| --- |
| The code stores a string literal in a char\* and then attempts to change it which often results in a crash |
| char \*str = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| This code uses a string literal and specifies the initial value in an array and the size of the array so it can be modified safely. |
| char str[] = "string literal";  str[0] = 'S'; |

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

| **Principles(s):** 4. Keep It Simple  Using simple methods for strings prevent bugs while modifying read-only memory.  9. Use Effective Quality Assurance Techniques  QA is used to find modifications of string literals and detect errors early on to prevent a crash. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium - leads to a program crash | Unlikely - this usually shows up when the program is being compiled | Low - usually easy to fix with the compiler error notification | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang++ | 19.1.0 | Warning\_str\_modify\_literal | Clang++ or Clang detects any attempts to modify the memory and show any mismatches. |

#### Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | FIO47-C. Use valid format strings  Invalid strings can lead to program crashes or incorrect input, which can cause security risks. This can happen when the format does not match the argument so following this rule ensures that functions work correctly.  https://wiki.sei.cmu.edu/confluence/display/c/FIO47-C.+Use+valid+format+strings |

| **Noncompliant Code** |
| --- |
| This code mismatches the arguments and conversion specifics which results in an error |
| 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** |
| --- |
| This code makes sure that the arguments to the print function to match the conversion specifics |
| 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):** 7. Sanitize Data Sent to Other Systems  Check that the format strings are valid and prevent injection attacks.  1.Validate Input Data  Input validation checks that the input data is as expected and prevents a crash. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High - can lead to vulnerabilities in the code. | Unlikely | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify | 22.2.x | Format\_string\_checks | Checks that the string arguments match to prevent errors or points that can be exploited. |

#### Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | MEM30-C. Do not access freed memory  Accessing memory after it has been freed can cause unexpected behavior and lead to corrupted data and security risks. Following the rule ensures that memory is only accessed when it should be.  https://wiki.sei.cmu.edu/confluence/display/c/MEM30-C.+Do+not+access+freed+memory |

| **Noncompliant Code** |
| --- |
| This code uses an incorrect technique to free memory in a linked list |
| 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** |
| --- |
| This code stores a reference to p->next in q before freeing memory p |
| 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  QA testing that covers a variety of scenarios will catch any invalid access before it corrupts the data.  8. Practice Defense in Depth  There should be multiple memory checks to live up to safe coding practices and prevent exploitation. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High - may lead to code injection | Unlikely | Medium - this can be time consuming | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| GCC | 15.1 | mem\_access\_after\_free | GCC uses dynamic memory to detect invalid dereferencing after the deallocation. |

#### Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | MSC37-C. Ensure that control never reaches the end of a non-void function  Making sure the control flow never gets to the end of a non-void function without returning a value is important to prevent random behavior. Checking the control paths will return a valid value to prevent program crashes.  https://wiki.sei.cmu.edu/confluence/display/c/MSC37-C.+Ensure+that+control+never+reaches+the+end+of+a+non-void+function |

| **Noncompliant Code** |
| --- |
| This code has a control that reaches the end of the function where the two strings are not equal, causing an error. |
| int checkpass(const char \*password) {  if (strcmp(password, "pass") == 0) {  return 1;  }  }    void func(const char \*userinput) {  if (checkpass(userinput)) {  printf("Success\n");  }  } |

| **Compliant Code** |
| --- |
| This code uses a function that always returns a value. |
| int checkpass(const char \*password) {  if (strcmp(password, "pass") == 0) {  return 1;  }  return 0;  }    void func(const char \*userinput) {  if (checkpass(userinput)) {  printf("Success!\n");  }  } |

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

| **Principles(s):** 2. Heed Compiler Warnings  Compilers give warnings that a function has no return, which will help to address warnings early on in development.  4.Keep It Simple  Functions with clear behavior checks that each path returns a value which avoids bugs. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely | Low | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.16.0 | Control\_flow, missing\_return | Cppcheck finds functions that don’t have return statements or unfound paths. |

Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | ERR33-C. Detect and handle standard library errors  Library errors that are not checked or handled can lead to program crashes and security risks. Library functions send out errors that need to be specifically handled, so following this rule ensure the program is secure.  https://wiki.sei.cmu.edu/confluence/display/c/ERR33-C.+Detect+and+handle+standard+library+errors |

| **Noncompliant Code** |
| --- |
| This code converts UTF-8 characters to wide characters, but does not check for an error in the setlocale() function which can cause a failure that is not properly handled. |
| 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** |
| --- |
| This code checks that the value returned by setlocale() is correct and restores the settings before returning the control. |
| 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):** 8. Practice Defense in Depth  If you handle library level errors, you can add in a safety buffer to avoid a third-party code fail.  9. Use Effective Quality Assurance Techniques  Testing checks that library functions work as they should and errors are handled. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Likely - common mistake | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang++ | 19.1.0 | Unhandled\_errors, library\_call\_functions | Clang++ gives a warning when no values are returned or they are not checked after a library call. |

#### Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Code Quality | [STD-008-CPP] | MSC41-C. Never hard code sensitive information  Hard-coding authentication or sensitive information risks exposure and unauthorized access to malignant users. By keeping this type of information outside of the code, it improves security and prevents leaks.  https://wiki.sei.cmu.edu/confluence/display/c/MSC41-C.+Never+hard+code+sensitive+information |

| **Noncompliant Code** |
| --- |
| This code authenticates to a remote service using a function, but passes the authentication to the function as a string and makes it easy to access to malignant users. |
| /\* Returns nonzero if authenticated \*/  int authenticate(const char\* code);    int main() {  if (!authenticate("correct code")) {  printf("Authentication error\n");  return -1;  }    printf("Authentication successful\n");  // ...Work with system...  return 0;  } |

| **Compliant Code** |
| --- |
| This code requires the user to input the authentication and erases it at the end so it is more secure. |
| /\* 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\_explicit(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):** 5. Default Deny  By making sure that the default is to deny access, you can verify credentials reducing exposure.  3. Architect and Design for Security Policies  Avoid using hardcoded secrets is in line with security best practices so that credentials are centralized in the system and include encryption. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Low | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| TruffleHog | v3.90.1 | Hardcoded\_secret, credential exposure | TruffleHog scans for keywords to detect any hardcoded secrets. |

Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### 

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Validation | [STD-009-CPP] | FIO30-C. Exclude user input from format strings  Including user input in format strings causes vulnerabilities to the system and causes it to crash. By avoiding this, the system is protected and becomes more stable overall.  https://wiki.sei.cmu.edu/confluence/display/c/FIO30-C.+Exclude+user+input+from+format+strings |

| **Noncompliant Code** |
| --- |
| This code accepts the name of a user as a string and displays an error if the user and password is not found, but contains a security risk since the msg includes input that cannot be trusted. |
| 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 code uses a call to fputs() rather than fprintf() which outputs the msg directly without checking the contents so it its not a security risk. |
| 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):** 7. Sanitize Data Sent to Other Systems  Removing user input from the strings saves the system from injection attacks and verifies user output is safely handled.  1. Validate Input Data  Checking that user input doesn’t become a part of the string protects the system from vulnerabilities. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High - vulnerable to formatted string attacks | Likely | Low | High | 4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.16.0 | format\_string\_injection | Finds improper user input in formatted functions. |

Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

#### 

#### 

#### 

#### 

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | CON37-C. Do not call signal() in a multithreaded program  Using signals in multithread calls causes unexpected behavior in the code since signals can be delivered unexpectedly. By avoiding using signals, the program is safer and handles threads better.  https://wiki.sei.cmu.edu/confluence/display/c/CON37-C.+Do+not+call+signal%28%29+in+a+multithreaded+program |

| **Noncompliant Code** |
| --- |
| This code uses a signal function from a multithread program which goes against the standard. |
| volatile sig\_atomic\_t flag = 0;    void handler(int signum) {  flag = 1;  }    /\* Runs until user sends SIGUSR1 \*/  int func(void \*data) {  while (!flag) {  /\* ... \*/  }  return 0;  }    int main(void) {  signal(SIGUSR1, handler); /\* Undefined behavior \*/  thrd\_t tid;    if (thrd\_success != thrd\_create(&tid, func, NULL)) {  /\* Handle error \*/  }  /\* ... \*/  return 0;  } |

| **Compliant Code** |
| --- |
| This code uses an object type so the child thread knows to terminate the loop and complies with the coding standard. |
| atomic\_bool flag = ATOMIC\_VAR\_INIT(false);    int func(void \*data) {  while (!flag) {  /\* ... \*/  }  return 0;  }    int main(void) {  thrd\_t tid;    if (thrd\_success != thrd\_create(&tid, func, NULL)) {  /\* Handle error \*/  }  /\* ... \*/  /\* Set flag when done \*/  flag = true;    return 0;  } |

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

| **Principles(s):** 4. Keep It Simple  Avoiding signals in environments keeps the program clear and easier to debug for other developers.  9. Use Effective Quality Assurance Techniques  QA helps catch signal misbehaviour early on in the process, which is helpful when multi-threading is involved. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Medium | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.16.0 | thread\_signal\_interaction | This will detect unsafe signal use. |

Software Engineering Institute. (n.d.). *SEI CERT C++ Coding Standard*. Retrieved July 11, 2025, from<https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>

### 

### 

### 

### 

### 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.

It’s important to include security in the DevOps pipeline to make sure the highest coding standards are followed and that they are consistent. Each of the stages should include automated checks starting with:

To enforce the coding standards and make sure security is applied across development, I need to use automated tools like Cppcheck and Fortify. The given diagram shows how security checks need to be incorporated throughout the process, not just at the end. By including security automation at each phase, I can ensure consistent compliance, early detection of any issues that might exist and lower the cost of fixes. By incorporating automation tools and checks, the program will be more secure and reliable.

* Assess and Plan: During this phase, secure coding policies should be decided on and tools should be chosen. The SEI CERT guidelines should be adhered to and a tool like Cppcheck can be used to check for any policy violations.
* Design: Security should be built directly into the design so memory handling, input validation, and format string safety should all be checked. Adding in secure design templates can ensure default deny and least privilege are incorporated from the start.
* Build: As the code is being written, static tools like Cppcheck or Clang++ should be run automatically during the building process. Those tools can find any violations with the rules like overflows (INT35-C) or hardcoded secrets (MSC41-C). This will provide immediate feedback to find any errors.
* Verify and Test: Automated tests like unit testing should be run throughout the process and with every change to the code. This will prevent any logic errors or improper conversions (INT31-C). Test automation tools like GoogleTest can be used as well to check that real-world conditions are used.
* Transition and Health Check: Before going live with the code, more scans should be completed using something like Fortify to check for any hardcoded credentials or areas that could expose data. This should also include any final policy checks to make sure the code is meeting all the guidelines laid out.
* Monitor and Detect, Respond: Once the code is live, monitoring tools like ELK Stack should be used to find any strange behavior. If any policy violations or attacks are found, the tools should alert the developer so fixes can be made. This will allow errors to be found and corrected in a timely manner.
* Maintain and Stabilize: As new threats are discovered or standards are updated, automation tools should be updated so that they follow the latest best practices.

### 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 | Likely | Low | High | 4 |
| STD-002-CPP | High | Likely | Low | High | 4 |
| STD-003-CPP | Medium | Unlikely | Low | Medium | 3 |
| STD-004-CPP | High | Unlikely | Low | High | 4 |
| STD-005-CPP | High | Unlikely | Medium | High | 4 |
| STD-006-CPP | Medium | Likely | Low | Medium | 3 |
| STD-007-CPP | Medium | Likely | Low | High | 4 |
| STD-008-CPP | High | Likely | Low | High | 5 |
| STD-009-CPP | High | Likely | Low | High | 4 |
| STD-010-CPP | Medium | Unlikely | Medium | Medium | 3 |

### 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 | What it is: Encryption at rest is the process of protecting stored data in a physical format like a hard or cloud drive by encrypting it. This will protect the data that is not actively moving throughout the network.  Policy: All sensitive data should be encrypted at rest using SHA-256. Database encryption must be applied to all company-managed or cloud-hosted systems.  How and why it Applies: If a device is stolen or compromised in any way, the encryption at rest will protect the data. This will protect against theft or authorized access to the system. |
| Encryption in flight | What it is: Encryption in flight (or in transit) refers to protecting data that is being moved across networks like between client and server data. This will prevent the data from being compromised during transit.  Policy: All data transmitted on either public or private networks needs to be encrypted using HTTPS protocols. Secure channels like VPN should also be used for added protection.  How and why it Applies: Data in transit is particularly vulnerable to attacks so encryption in flight to protect data and the success of the transmission. This will apply to user logins or third party access. |
| Encryption in use | What it is: Encryption in flight protects data while it is actively being processed in memory so it needs to be protected using trusted execution environments.  Policy: For sensitive data that should have added protection like user information, should all be processed in a secure environment. Using trusted execution environments is important when working in a cloud transmission.  How and why it Applies: This should be used to protect against attackers who might try to access the memory or exploit the open channels. The encryption in use will protect against these threats while it’s running, particularly in untrusted environments. |

Djalovic, N. (2025, April 8). A comprehensive guide to data encryption: Data at rest, in motion, and in use. Jatheon. https://jatheon.com/blog/data-at-rest-data-in-motion-data-in-use/

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | What it is: Authentication is the process that verifies a user or system to ensure they should have access to the system. This will ensure the identity is verified before they are granted access.  Policy: All logins must include strong authentication using multi-factor authentication for admin access and passwords must follow established guidelines.  How and why it Applies: This will prevent unauthorized users from being granted access to the system and will not allow the identity to be faked. The credentials will be protected and only those who should be in the system will have access. |
| Authorization | What it is: Authorization is the process of deciding what each authenticated user is allowed to do within the system. This will determine the roles and what level of access they should have.  Policy: Role-Based access should be implemented across the system so users will have the least amount of privilege they need to complete their work. Any changes to the user roles or access must be approved.  How and why it Applies: This will prevent privilege escalation, unauthorized access from accounts that have been hacked, and ensure the access by role is clear. |
| Accounting | What it is: Accounting is the process of tracking the user activities within the system or any changes. This will provide a much needed record of which roles did which actions at all times of day.  Policy: All logins, access, and changes in the database should be tracked and logged in the system. The logs should then be stored securely in a separate location and closely monitored for any irregularities.  How and why it Applies: This will support reporting for any incidents and records for audits that will be required for investigation. This policy will make sure that any irregularities is found, tracked, and identified. |

Fortinet Inc. (n.d.). What is AAA security? In CyberGlossary. Fortinet. Retrieved July 31, 2025, from <https://www.fortinet.com/resources/cyberglossary/aaa-security>

**\***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 | 07/15/2025 | Updated template for Module Four | Danielle McNeill |  |
| 3.0 | 08/06/2025 | Revised template for Project One | Danielle McNeill |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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