**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Validating input data is a vital part of secure coding. By validating the data, you will help protect yourself from buffer overflow/underflow which can allow an attacker to gain unauthorized access to private areas of the program or gain privileged information. |
| 1. Heed Compiler Warnings | Compilers come with a lot of warnings that help identify possible vulnerabilities, it is important to not ignore them. While the compiler may not catch everything, and it is important to code securely from the start, the warning list is a useful starting point for ensuring the code is secure. |
| 1. Architect and Design for Security Policies | One of the best ways to ensure a program is coded securely is to keep security policies and best practices in mind during the entire software development lifecycle. Do not be reactionary with secure coding. By using secure coding practices from inception through release, the company will save time, money, and public relations. |
| 1. Keep It Simple | One of the biggest coding principles is to keep the program as simple as possible. Avoid over complicated programs, ensure that names of variables and functions are descriptive, and be sure to use comments throughout the code to illustrate intention and explanations. By doing this you can reduce errors both in coding and in use. |
| 1. Default Deny | When creating access to secure areas of the program, deny permissions by default. The only way that access should be given is when specific conditions are met. By denying access by default, another layer of protection is added that can prevent outsider access to secure areas of the program. |
| 1. Adhere to the Principle of Least Privilege | When designing levels of privilege, users should be given as little access as possible. By only giving the access that is necessary, you can help protect yourself from unauthorized access to privilaged areas of the software. |
| 1. Sanitize Data Sent to Other Systems | By sanitizing the data, you can help protect yourself from things like SQL attacks and invalid input and output data. |
| 1. Practice Defense in Depth | When designing your cyber security, use a series of independent layers of security so that even if a single layer is made vulnerable, the attack will not be successful. |
| 1. Use Effective Quality Assurance Techniques | It is important to test the security you have developed in your program. Auditing, penetration testing, and bound testing are vital to ensure that the program is as secure as possible. It is not enough to assume you have accounted for all possible vulnerabilities. |
| 1. Adopt a Secure Coding Standard | Apply the best secure coding practices to your code from the beginning and in every program that you work on. |

### 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: Select the Right Data Type** |
| --- | --- | --- |
| **Data Type** | [STD-001-CPP] | Selecting a data type that suits the program's needs and can accommodate all likely valid inputs allows for the secure handling of the data and proper coding and use of the program. |

Source: <https://wiki.sei.cmu.edu/confluence/display/java/NUM03-J.+Use+integer+types+that+can+fully+represent+the+possible+range+of++unsigned+data>

| **Noncompliant Code** |
| --- |
| An inventory management system that checks the number of apples in the stores inventory.  The number of apples will never be negative, having the variable be signed cuts the range in half needlessly.  Since the number of apples in a single store's inventory is unlikely to be above 65,535 it is an unnecessary use of memory to use int as the data type for both variables.  There is no check to ensure the input data is a valid answer. |
| int main() {  signed int numAppleInv;  static int APPLESTOCKMIN = 5;    std::cout << "How many apples are in stock?" << std::endl;  std::cin >> numAppleInv;    if (numAppleInv < APPLESTOCKMIN) {  std::cout << "We need to order more apples." << std::endl;  }  else {  std::cout << "We do not need to order more apples." << std::endl;  }    return 0;  } |

| **Compliant Code** |
| --- |
| An inventory management system that checks the number of apples in the stores inventory.  The number of apples will never be negative, having the variable be unsigned gives a larger capacity without sacrificing any memory.  Since the number of apples in a single store's inventory is unlikely to be above 65,535 using the short int saves on memory and improves performance.  The input is checked to ensure that it is within the range of short int, if it is not then an exception is thrown. |
| int main() {  unsigned short int numAppleInv;  static unsigned short int APPLESTOCKMIN = 5;  bool inputValidated = false;    while (!inputValidated) {  std::cout << "How many apples are in stock? (0-65,000)"  << std::endl;  try {  std::cin >> numAppleInv;  }  catch (const std::overflow\_error& error) {  std::cout << error.what();  std::cout << "How many apples are in stock? (0-65,000)"  << std::endl;  std::cin.ignore(INT\_MAX);  std::cin >> numAppleInv;  }    if ((numAppleInv > 65000) || (numAppleInv < 0)) {  throw std::overflow\_error("You entered an invalid  response.\n");  }  else {  inputValidated = true;  }  }    if (numAppleInv < APPLESTOCKMIN) {  std::cout << "We need to order more apples." << std::endl;  }  else {  std::cout << "We do not need to order more apples."  << std::endl;  }    return 0;  } |
|  |

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

| **Principles(s):**   * Validate Input Data: Always validate all information received from an outside source. * Architect and Design for Security policies – Automated detection for this is infeasible, so preventing it from the start is vital. * Keep It Simple – Choosing the smallest data type that can support all valid inputs and choosing it from the start saves the system resources and the company time and money. * Practice Defense in Depth – Using the proper variable type is the first layer in the defense against vulnerabilities like over and underflow errors. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | Medium | 4 |

**Automation**

|  |
| --- |
| Automated detection is infeasible in the general case. If security is kept in mind throughout the development lifecycle however, this vulnerability should not be a problem. Ensure that data is validated and that the proper variable types are used. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard: Ensure Integer Conversions Do Not Result in Loss of Data** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | When converting from one type of integer to another, it is vital that there is no loss of data, and that the data is not misinterpreted. This is especially important when dealing with the character data type. |

source: <https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data>

| **Noncompliant Code** |
| --- |
| An inventory management system takes in the number of apples purchased and orders more to ensure the store has a constant inventory of 40000 apples.  There is a conversion of unsigned to sign resulting in an error if the purchased amount is above the max range of the signed int.  Only having one arithmetic function for handling purchasing limits the number of apples to a max of only being able to sell what is in the store  The number of apples to purchase is the short int along with the other variables, even though it could go above the max and cause an overflow error |
| int main() {  unsigned short int numAppleInv = 40000;  signed short int numAppleSale;  signed short int numAppleToPurchase;    std::cout << "How many apples are being purchased?\n" << std::endl;  std::cin >> numAppleSale;    numAppleToPurchase = numAppleInv - numAppleSale;  std::cout << "Okay, ordering " << numAppleToPurchase << " apples." << std::endl;    return 0;  } |

| **Compliant Code** |
| --- |
| An inventory management system takes in the number of apples purchased and orders more to ensure the store has a constant inventory of 40000 apples.  All the variables for the apple inventory have a matching sign, and there is a validation for the user input to ensure that the number of apples are within range  There is an arithmetic function for ordering less than the store inventory and more than the store inventory, allowing for the purchase of more apples and the system will still replenish up to the 40000  The variable for the amount of apples to purchase is now an int instead of a short int to accommodate the maximum apples purchasable (65000 for the short int max + 40000 for the store inv) |
| int main() {  unsigned short int numAppleInv = 40000;  unsigned short int numAppleSale;  unsigned int numAppleToPurchase;  bool inputValidated = false;    while (!inputValidated) {  std::cout << "How many apples are being purchased?\n" <<  std::endl;  try {  std::cin >> numAppleSale;  }  catch(const std::overflow\_error& error){  std::cout << error.what();  std::cout << "How many apples are being purchased?\n" <<  std::endl;  }    if (0 > numAppleSale > 65000) {  throw std::overflow\_error("You entered an invalid  response.\n");  }  else {  inputValidated = true;  if (numAppleInv > numAppleSale) {  numAppleToPurchase = numAppleInv - numAppleSale;  std::cout << "Okay, ordering " << numAppleToPurchase  << " apples." << std::endl;  }  else {  numAppleToPurchase = (numAppleSale - numAppleInv) +  40000;  std::cout << "Okay, ordering " << numAppleToPurchase  << " apples." << std::endl;  }  }  }  return 0;  } |

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

| **Principles(s):**   * Validate Input Data: When taking in data from a user, ensure that their input will not invalidate how the system operates (if the input should be an int, don’t allow a char to be used). * Heed Compiler Warnings: These errors are often caught by compilers, making sure that warning and errors are heeded can prevent problems before they exist. * Sanitize Data Sent to Other Systems: It is important to sanitize data both coming in and going out of the system to make sure that no data is being lost or misrepresented. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | High | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | 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 |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect violations of this rule. However, false warnings may be raised if limits.h is included |
| [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 |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | C2850, C2855, C2890, C2895, C2900, C2905,    C++2850, C++2855, C++2890, C++2895, C++2900, C++2905, C++3000, C++3010        DF2851, DF2852, DF2853, DF2856, DF2857, DF2858, DF2891, DF2892, DF2893, DF2896, DF2897, DF2898, DF2901, DF2902, DF2903, DF2906, DF2907, DF2908 |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | PORTING.CAST.SIZE |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 93 S, 433 S, 434 S | Partially implemented |
| Parasoft C/C++test | 2023.1 | 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 | An expression of essentially Boolean type should always be used where an operand is interpreted as a Boolean value  An operand of essentially Boolean type should not be used where an operand is interpreted as a numeric value  An operand of essentially character type should not be used where an operand is interpreted as a numeric value  An operand of essentially enum type should not be used in an arithmetic operation  Shift and bitwise operations should not be performed on operands of essentially signed or enum type  An operand of essentially signed or enum type should not be used as the right hand operand to the bitwise shifting operator  An operand of essentially unsigned type should not be used as the operand to the unary minus operator  The value of an expression shall not be assigned to an object with a narrower essential type  The value of an expression shall not be assigned to an object of a different essential type category  Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category  The second and third operands of the ternary operator shall have the same essential type category  The value of a composite expression shall not be assigned to an object with wider essential type  If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type  If a composite expression is used as one (second or third) operand of a conditional operator then the other operand shall not have wider essential type  Avoid data loss when converting between integer types  Avoid value change when converting between integer types |
| Polyspace Bug Finder | R2024a | [CERT C: Rule INT31-C](https://www.mathworks.com/help/bugfinder/ref/certcruleint31c.html) | Checks for:   * Integer conversion overflow * Call to memset with unintended value * Sign change integer conversion overflow * Tainted sign change conversion * Unsigned integer conversion overflow   Rule partially covered. |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.30 | [V562](https://pvs-studio.com/en/docs/warnings/v562/), [V569](https://pvs-studio.com/en/docs/warnings/v569/), [V642](https://pvs-studio.com/en/docs/warnings/v642/), [V676](https://pvs-studio.com/en/docs/warnings/v676/), [V716](https://pvs-studio.com/en/docs/warnings/v716/), [V721](https://pvs-studio.com/en/docs/warnings/v721/), [V724](https://pvs-studio.com/en/docs/warnings/v724/), [V732](https://pvs-studio.com/en/docs/warnings/v732/), [V739](https://pvs-studio.com/en/docs/warnings/v739/), [V784](https://pvs-studio.com/en/docs/warnings/v784/), [V793](https://pvs-studio.com/en/docs/warnings/v793/), [V1019](https://pvs-studio.com/en/docs/warnings/v1019/), [V1029](https://pvs-studio.com/en/docs/warnings/v1029/), [V1046](https://pvs-studio.com/en/docs/warnings/v1046/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6 and 10.7 |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | signed\_downcast | Exhaustively verified. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard: Detect Errors when Converting a String to a Number** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | When converting from a string to a number there can be several errors. The string numbers could be out of range of the number, the string could contain non number symbols, etc. It is important to make sure you verify the string before the conversion occurs. |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR62-CPP.+Detect+errors+when+converting+a+string+to+a+number>

| **Noncompliant Code** |
| --- |
| An inventory management system takes in the number of apples purchased and orders more to ensure the store has a constant inventory of 40000 apples.  Since there is no check to ensure that the input is both a number and can be stored in a short int, the system ignores the input but does not file the exception. Instead, the system orders 4000 more apples even though an invalid input was entered. |
| int main() {  unsigned short int numAppleInv = 40000;  unsigned short int numAppleSale;  unsigned int numAppleToPurchase;  bool inputValidated = false;    while (!inputValidated) {  std::cout << "How many apples are being purchased?\n" <<  std::endl;  try {  std::cin >> numAppleSale;  }  catch(const std::overflow\_error& error){  std::cout << error.what();  std::cout << "How many apples are being purchased?\n" <<  std::endl;  }    if (0 > numAppleSale > 65000) {  throw std::overflow\_error("You entered an invalid  response.\n");  }  else {  inputValidated = true;  if (numAppleInv > numAppleSale) {  numAppleToPurchase = numAppleInv - numAppleSale;  std::cout << "Okay, ordering " << numAppleToPurchase  << " apples." << std::endl;  }  else {  numAppleToPurchase = (numAppleSale - numAppleInv) +  40000;  std::cout << "Okay, ordering " << numAppleToPurchase  << " apples." << std::endl;  }  }  }  return 0;  } |

| **Compliant Code** |
| --- |
| An inventory management system takes in the number of apples purchased and orders more to ensure the store has a constant inventory of 40000 apples.  The input stream is checked to ensure it fits within a short int variable. There is a new catch to handle the exception and the if statements for the calculations now include the 2 intended paths and a catchall for any other path. |
| int main() {  unsigned short int numAppleInv = 40000;  unsigned short int numAppleSale;  unsigned int numAppleToPurchase;  bool inputValidated = false;    while (!inputValidated) {  std::cout << "How many apples are being purchased?\n" <<  std::endl;  std::cin.exceptions(std::istream::failbit |  std::istream::badbit);  try {  std::cin >> numAppleSale;  }  catch(const std::overflow\_error& error){  std::cout << error.what();  return 0;  }  catch (std::istream::failure& error) {  std::cout << error.what();  return 0;  }    if (0 > numAppleSale > 65000) {  throw std::overflow\_error("You entered an invalid  response.\n");  }  else {  inputValidated = true;  if (numAppleInv > numAppleSale) {  numAppleToPurchase = numAppleInv - numAppleSale;  std::cout << "Okay, ordering " << numAppleToPurchase  << " apples." << std::endl;  }  else if (numAppleInv <= numAppleSale){  numAppleToPurchase = (numAppleSale - numAppleInv) +  40000;  std::cout << "Okay, ordering " << numAppleToPurchase  << " apples." << std::endl;  }  else {  std::cout << "Error in processing, please try again."  << std::endl;  }  }  }  return 0;  } |

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

| **Principles(s):**   * ValidateInput Data: While taking user input as a string can avoid a lot of errors in preventing crashes, it is still vital that that string is validated BEFORE conversion to a different type of data to ensure correctness. * Architect and Design for Security Policies: Keeping track of every variable can be incredibly difficult when a project gets into its final stage, keeping in mind string conversions and validating them as the project progresses ensures that nothing gets missed. * Sanitize Data Sent to Other Systems: Sending incorrect data types can quickly break a program, so sanitize all data. * Practice Defense in Depth: It is not enough to just check if an input is within a range, it is also important to ensure that the data is also the right type of data. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR62** | [Insert text.] |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang) | 3.9 | cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 8.1p0 | **BADFUNC.ATOF** **BADFUNC.ATOI** **BADFUNC.ATOL** **BADFUNC.ATOLL** | Use of atof Use of atoi Use of atol Use of atoll |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++3161** | [Insert text.] |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **CERT.ERR.CONV.STR\_TO\_NUM** |  |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2023.1 | **CERT\_CPP-ERR62-a** | The library functions atof, atoi and atol from library stdlib.h shall not be used |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2024a | [CERT C++: ERR62-CPP](https://www.mathworks.com/help/bugfinder/ref/certcerr62cpp.html) | Checks for unvalidated string-to-number conversion (rule fully covered) |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard: Sanitize Data Passed to Complex Subsystems** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | String data that is passed to a complex subsystem may contain special characters that can trigger commands or actions. This can result in unpredicted behavior at best and unauthorized access to sensitive data at worst. It is necessary to always sanitize the data before passing it to ensure protection from things like SQL attacks. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems>

| **Noncompliant Code** |
| --- |
| Inputs an email address to a buffer and then uses this string as an argument in a call to system() (Seacord, 2022). |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| Acceptable data is allowed while unacceptable data is sanatized. One way to accomplish this is by whitelisting the approved characters (Seacord, 2022). |
| 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):**   * Validate Input Data: Always ensure that all data coming from an untrusted source is validated to prevent attacks * Keep It Simple: Ensure that input is validated in an easy to read and understand fashion so that nothing is forgotten and the security is easy to maintain * Sanitize Data Sent to Other Systems: Always protect against things like SQL attacks by sanitizing data before sending to other systems |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported by stubbing/taint analysis |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **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](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 6.5 | **TAINTED\_STRING** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **NNTS.TAINTED** **SV.TAINTED.INJECTION** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **108 D, 109 D** | Partially implemented |
| [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) | R2024a | [CERT C: Rec. STR02-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.str02c.html) | 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: Do Not Read Uninitialized Memory** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Ensure that all variables are initialized before being called. Just declaring them leaves the values as indeterminate and unpredictable. Also, some memory allocation functions do not initialize the memory allocated for that variable. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/EXP33-C.+Do+not+read+uninitialized+memory>

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the set\_flag() function is intended to set the parameter, sign\_flag, to the sign of number. However, the programmer neglected to account for the case where number is equal to 0. Because the local variable sign is uninitialized when calling set\_flag() and is never written to by set\_flag(), the comparison operation exhibits undefined behavior when reading sign (Gennari, 2023). |
| void set\_flag(int number, int \*sign\_flag) {  if (NULL == sign\_flag) {  return;  }    if (number > 0) {  \*sign\_flag = 1;  } else if (number < 0) {  \*sign\_flag = -1;  }  }    int is\_negative(int number) {  int sign;  set\_flag(number, &sign);  return sign < 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution trivially repairs the problem by accounting for the possibility that number can be equal to 0 (Gennari, 2023).  An additional defense-in-depth practice worth considering is to initialize local variables immediately after declaration (Gennari, 2023). |
| void set\_flag(int number, int \*sign\_flag) {  if (NULL == sign\_flag) {  return;  }    /\* Account for number being 0 \*/  if (number >= 0) {  \*sign\_flag = 1;  } else {  \*sign\_flag = -1;  }  }    int is\_negative(int number) {  int sign = 0; /\* Initialize for defense-in-depth \*/  set\_flag(number, &sign);  return sign < 0;  } |

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

| **Principles(s):**   * Architect and Design for Security Policies: Ensure that variables are initialized throughout the lifecycle of the system, waiting until the end to check every variable is infeasible and wastes both time and money. * Adopt a Secure Coding Standard: Using secure variables means that they are initialized from the beginning of their use through them no longer being needed. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Probable | Medium | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **uninitialized-local-read**  **uninitialized-variable-use** | Fully checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-EXP33** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.MEM.UVAR** | Uninitialized variable |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Automatically detects simple violations of this rule, although it may return some false positives. It may not catch more complex violations, such as initialization within functions taking uninitialized variables as arguments. It does catch the second noncompliant code example, and can be extended to catch the first as well |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | **UNINIT** | Implemented |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | **uninitvar** **uninitdata** **uninitstring** **uninitMemberVar** **uninitStructMember** | Detects uninitialized variables, uninitialized pointers, uninitialized struct members, and uninitialized array elements (However, if one element is initialized, then cppcheck assumes the array is initialized.) There are FN compared to some other tools because Cppcheck tries to avoid FP in impossible paths. |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 |  | Can detect some violations of this rule when the -Wuninitialized flag is used |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **DF2726, DF2727, DF2728, DF2961, DF2962, DF2963, DF2966, DF2967, DF2968, DF2971, DF2972, DF2973, DF2976, DF2977, DF2978** | Fully implemented |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **UNINIT.HEAP.MIGHT** **UNINIT.HEAP.MUST** **UNINIT.STACK.ARRAY.MIGHT** **UNINIT.STACK.ARRAY.MUST** **UNINIT.STACK.ARRAY.PARTIAL.MUST** **UNINIT.STACK.MIGHT** **UNINIT.STACK.MUST** | Fully implemented |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **53 D, 69 D, 631 S, 652 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-EXP33-a** | Avoid use before initialization |
| [Parasoft Insure++](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 |  | Runtime analysis |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **530, 603, 644, 901** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rule EXP33-C](https://www.mathworks.com/help/bugfinder/ref/certcruleexp33c.html) | Checks for:   * Non-initialized variable * Non-initialized pointer   Rule partially covered |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.30 | [**V573**](https://pvs-studio.com/en/docs/warnings/v573/), [**V614**](https://pvs-studio.com/en/docs/warnings/v614/), [**V670**](https://pvs-studio.com/en/docs/warnings/v670/), [**V679**](https://pvs-studio.com/en/docs/warnings/v679/), [**V1050**](https://pvs-studio.com/en/docs/warnings/v1050/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **uninitialized-local-read** | Partially checked |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/c/Splint) | 3.1.1 |  |  |
| [TrustInSoft Analyzer](https://wiki.sei.cmu.edu/confluence/display/c/TrustInSoft+Analyzer) | 1.38 | **initialisation** | Exhaustively verified (see [one compliant and one non-compliant example](https://taas.trust-in-soft.com/tsnippet/t/c72964e2)). |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard: Use a Static Assertion to Test the Value of a Constant Expression** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Assertions are a valuable tool in helping to find vulnerabilities in code. However, because of the runtime overhead and because it calls abort(), it should only be used for identifying incorrect assumptions and not for runtime error checking. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression>

| **Noncompliant Code** |
| --- |
| This noncompliant code uses the assert() macro to assert a property concerning a memory-mapped structure that is essential for the code to behave correctly (Seacord & Britton, 2018). |
| #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** |
| --- |
| For assertions involving only constant expressions, a preprocessor conditional statement may be used, as in this compliant solution (Seacord & Britton, 2018). |
| 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):**   * Architect and Design for Security Policies: Do not use assertions as a reason to not code securely, they are to only be used for identifying incorrect assumptions. * Keep It Simple: Do not go overboard with assertions, they cost the system resources. Use only the assertions that you need. * Use Effective Quality Assurance Techniques: Perform regular tests to make sure that all assumptions are valid, and that no assumption is needlessly included. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL03** |  |
| [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) | 8.1p0 | **(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 |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **44 S** | Fully implemented |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard: Handle All Exceptions** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | When an exception is thrown, if there is no handler found within the handler's try block, the search for a handler will search other try blocks within the same thread. If there is no handler, then the default will be used. The default std::terminate() calls std::abort() which terminated the process abnormally. When abort() is called the stack may not be properly unwound, destructors may not be called, and resources may be left in an indeterminate state. This improper handling of resources is a common foothold for a DoS attack. |

Source: <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR51-CPP.+Handle+all+exceptions>

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called (Ballman, 2023). |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |

| **Compliant Code** |
| --- |
| In this compliant solution, the main entry point handles all exceptions, which ensures that the stack is unwound up to the main() function and allows for graceful management of external resources (Ballman, 2023). |
| 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):**   * Architect and Design for Security Policies: When coding, including the exception handlers from the start, trying to go back and cover every base is both difficult and wasteful. * Adopt a Secure Coding Standard: Including all exceptions will ensure that the default abort is not called and that all resources are deallocated properly. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 22.10 | **main-function-catch-all** **early-catch-all** | Partially checked |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC++-ERR51** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.STRUCT.UCTCH** | Unreachable Catch |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC) | 2024.1 | **C++4035, C++4036, C++4037** |  |
| [Klocwork](https://www.securecoding.cert.org/confluence/display/cplusplus/Klocwork) | 2024.1 | **MISRA.CATCH.ALL** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | **527 S** | Partially implemented |
| [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) | R2024a | [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: Use Comments Consistently and in a Readable Fashion** |
| --- | --- | --- |
| Comments | [STD-008-CPP] | When you are writing code, it is vital that comments are used throughout the code to ensure that it is maintainable. The comments must be consistent, relevant, readable, and understandable. If comments are omitted programs can quickly become unmaintainable as the code intentions become more complicated and less obvious to someone who did not write the source code. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/MSC04-C.+Use+comments+consistently+and+in+a+readable+fashion>

| **Noncompliant Code** |
| --- |
| Even though this is a very simple program and the variables are named in a descriptive manner, the intentions of the various if statements and exception handlers can be less obvious to another reader. Even in the simplest of programs comments should be used throughout. |
| int main() {  unsigned short int numAppleInv = 40000;  unsigned short int numAppleSale;  unsigned int numAppleToPurchase;  bool inputValidated = false;    while (!inputValidated) {  std::cout << "How many apples are being purchased?\n" <<  std::endl;  std::cin.exceptions(std::istream::failbit |  std::istream::badbit);  try {  std::cin >> numAppleSale;  }  catch(const std::overflow\_error& error){  std::cout << error.what();  return 0;  }  catch (std::istream::failure& error) {  std::cout << error.what();  return 0;  }    if (0 > numAppleSale > 65000) {  throw std::overflow\_error("You entered an invalid  response.\n");  }  else {  inputValidated = true;  if (numAppleInv > numAppleSale) {  numAppleToPurchase = numAppleInv - numAppleSale;  std::cout << "Okay, ordering " << numAppleToPurchase  << " apples." << std::endl;  }  else if (numAppleInv <= numAppleSale){  numAppleToPurchase = (numAppleSale - numAppleInv) +  40000;  std::cout << "Okay, ordering " << numAppleToPurchase  << " apples." << std::endl;  }  else {  std::cout << "Error in processing, please try again."  << std::endl;  }  }  }  return 0;  } |

| **Compliant Code** |
| --- |
| Adding comments throughout the code allows anyone reading the source code to understand the intended functionality, areas of code they may be unfamiliar with, and understand the reasoning behind certain design decisions. While it is not as apparent in this document because of the limited size available, “ // “ comments are to be used for comments taking up a single line and “ /\*...\*/ “ comments are to be used for comments taking up multiple lines. |
| int main() {  unsigned short int numAppleInv = 40000; // Number of apples in the  store inventory  unsigned short int numAppleSale; // Number of apples being sold by the store to the customer  // Number of apples needed to replenish the store stock, set to int to accomodate range of 2 \* short int  unsigned int numAppleToPurchase;  bool inputValidated = false; // Input validation    while (!inputValidated) {  // Asks user how many apples are being purchased  std::cout << "How many apples are being purchased?\n" << std::endl;  // Exception handler for user input that ensures input is an integer  std::cin.exceptions(std::istream::failbit | std::istream::badbit);    try { // Try to take integer from user for the number of apples being sold  std::cin >> numAppleSale;  }  catch(const std::overflow\_error& error){ // Exception for user input not being within range  std::cout << error.what();  return 0;  }  catch (std::istream::failure& error) { // Exception for input from user not being a. integer  std::cout << error.what();  return 0;  }    // If the inputed number is not within the given range then an exception is thrown  if (0 > numAppleSale > 65000) {  throw std::overflow\_error("You entered an invalid response.\n");  }  else {  inputValidated = true; // Input has been validated  /\*  \* If the number being sold is less than the store inventory then the system orders the number of apples  \* needed to replenish the store stock  \*/  if (numAppleInv > numAppleSale) {  numAppleToPurchase = numAppleInv - numAppleSale;  std::cout << "Okay, ordering " << numAppleToPurchase << " apples." << std::endl;  }  /\*  \* If the number being sold is greater than or equal to the stock then the math needs to accomodate that  \* and still replenish the store stock  \*/  else if (numAppleInv <= numAppleSale){  numAppleToPurchase = (numAppleSale - numAppleInv) + 40000;  std::cout << "Okay, ordering " << numAppleToPurchase << " apples." << std::endl;  }  else { // Catchall for error handling that falls outside of the intended use cases  std::cout << "Error in processing, please try again." << std::endl;  }  }  }  return 0;  } |

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

| **Principles(s):**   * Architect and Design for Security Policies: Making comments as you code ensures that nothing is left unknown to a future reader of the source code, be that future you or someone else * Keep It Simple: Keep comments simple, easy to understand, and consistent throughout your code |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | Low | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **mmline-comment** **sline-comment** **sline-splicing** **smline-comment** | Partially checked |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 |  | Can detect violations of this rule when the -Wcomment flag is used |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.MSC04** | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **C3108** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **119 S, 302 S, 611 S** | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MSC04-a** **CERT\_C-MSC04-b** **CERT\_C-MSC04-c** **CERT\_C-MSC04-d** | The character sequence /\* shall not be used within a C-style comment The character sequence // shall not be used within a C-style comment The character sequence /\* shall not be used within a C++-style comment Line-splicing shall not be used in // comments |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **1, 427, 602, 689, 853,** **9059, 9060, 9066, 9259** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rec. MSC04-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.msc04c.html) | Checks for use of /\* and // within a comment (rule partially covered) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **mmline-comment** **sline-comment** **sline-splicing** **smline-comment** | Partially checked |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard: Do Not Reuse Variable Names in Subscopes** |
| --- | --- | --- |
| Variable Naming | [STD-009-CPP] | Do not use the same variable name in two scopes where one scope is within another. Reusing names will lead to programmer confusion, reusing often means that the names of the variables are too generic and need to be adjusted, and could cause problems with the program not altering the correct variable. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/DCL01-C.+Do+not+reuse+variable+names+in+subscopes>

| **Noncompliant Code** |
| --- |
| This noncompliant code example declares the msg identifier at file scope and reuses the same identifier to declare a character array local to the report\_error() function. The programmer may unintentionally copy the function argument to the locally declared msg array within the report\_error() function. Depending on the programmer's intention, it either fails to initialize the global variable msg or allows the local msg buffer to overflow by using the global value msgsize as a bounds for the local buffer (Burch & Britton, 2023). |
| #include <stdio.h>    static char msg[100];  static const size\_t msgsize = sizeof( msg);    void report\_error(const char \*str) {  char msg[80];  snprintf(msg, msgsize, "Error: %s\n", str);  /\* ... \*/  }    int main(void) {  /\* ... \*/  report\_error("some error");    return 0;  } |

| **Compliant Code** |
| --- |
| This compliant solution uses different, more descriptive variable names (Burch & Britton, 2023). |
| #include <stdio.h>    static char message[100];  static const size\_t message\_size = sizeof( message);    void report\_error(const char \*str) {  char msg[80];  snprintf(msg, sizeof( msg), "Error: %s\n", str);  /\* ... \*/  }    int main(void) {  /\* ... \*/  report\_error("some error");    return 0;  } |

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

| **Principles(s):**   * Heed Compiler Warnings: Often this kind of issue can be prevented by heeding the warnings, errors, and messages in the compiler * Keep It Simple: Simple does not mean shortest. Variable names should be unique enough that reuse is not a factor. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 |  | Supported indirectly via MISRA C:2012 Rule 5.3. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 7.2.0 | **CertC-DCL01** |  |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 8.1p0 | **LANG.ID.ND.NEST** | Non-distinct identifiers: nested scope |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  |  |
| [ECLAIR](https://wiki.sei.cmu.edu/confluence/display/c/ECLAIR) | 1.2 | **CC2.DCL01** | Fully implemented |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **C0795, C0796, C2547, C3334** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **MISRA.VAR.HIDDEN** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **131 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-DCL01-a** **CERT\_C-DCL01-b** | Identifier declared in a local or function prototype scope shall not hide an identifier declared in a global or namespace scope Identifiers declared in an inner local scope should not hide identifiers declared in an outer local scope |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **578** | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rec. DCL01-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.dcl01c.html) | Checks for variable shadowing (rule fully covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.30 | [**V561**](https://pvs-studio.com/en/docs/warnings/v561/), [**V688**](https://pvs-studio.com/en/docs/warnings/v688/), [**V703**](https://pvs-studio.com/en/docs/warnings/v703/), [**V711**](https://pvs-studio.com/en/docs/warnings/v711/), [**V2015**](https://pvs-studio.com/en/docs/warnings/v2015/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 |  | Supported indirectly via MISRA C:2012 Rule 5.3. |
| [Splint](https://wiki.sei.cmu.edu/confluence/display/c/Splint) | 3.1.1 |  |  |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard: Strive for Logical Completeness** |
| --- | --- | --- |
| Logical Completeness | [STD-010-CPP] | Ensure that when you are writing your program that you consider all possible paths and all possible states of data. Failure to do so will result in vulnerabilities. |

Source: <https://wiki.sei.cmu.edu/confluence/display/c/MSC01-C.+Strive+for+logical+completeness>

| **Noncompliant Code** |
| --- |
| While the two main paths are considered where a is greater than or less than b, there is no path for if they are the same and the catchall will display an incorrect statement. |
| int main() {  signed int a = 14;  signed int b = 0;    std::cout << "Please enter a number." << std::endl;  std::cin >> b;    if (a < b) {  std::cout << "The number you entered is greater than " << a << std::endl  }  else {  std::cout << "The number you entered is less than " << a << std::endl  }  return 0;  } |

| **Compliant Code** |
| --- |
| By adding to the if statement, all possible states are accounted for, and the catchall will now only handle errors. |
| int main() {  signed int a = 14;  signed int b = 0;    std::cout << "Please enter a number." << std::endl;  std::cin >> b;    if (a < b) {  std::cout << "The number you entered is greater than " << a << std::endl  }  else if (a > b){  std::cout << "The number you entered is less than " << a << std::endl  }  else if (a == b) {  std::cout << "The number you entered is the same as " << a << std::endl  }  else {  std::cout << "I'm sorry, there seems to have been an error."<< std::endl  }  return 0;  } |

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

| **Principles(s):** [Name the principle and explain how it maps to this standard.] |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 24.04 | **missing-else**  **switch-default** | Partially checked |
| [Compass/ROSE](https://wiki.sei.cmu.edu/confluence/display/c/Rose) |  |  | Can detect some violations of this recommendation. In particular, it flags switch statements that do not have a default clause. ROSE should detect "fake switches" as well (that is, a chain of if statements each checking the value of the same variable). These if statements should always end in an else clause, or they should mathematically cover every possibility. For instance, consider the following:  if (x > 0) {  /\* ... \*/  } else if (x < 0) {  /\* ... \*/  } else if (x == 0) {  /\* ... \*/  } |
| [GCC](https://wiki.sei.cmu.edu/confluence/display/c/GCC) | 4.3.5 |  | Can detect some violations of this recommendation when the -Wswitch and -Wswitch-default flags are used |
| [Helix QAC](https://wiki.sei.cmu.edu/confluence/display/c/Helix+QAC) | 2024.1 | **C2000, C2002, C2004** |  |
| [Klocwork](https://wiki.sei.cmu.edu/confluence/display/c/Klocwork) | 2024.1 | **CWARN.EMPTY.LABEL**  **LA\_UNUSED** **MISRA.IF.NO\_ELSE** **MISRA.SWITCH.WELL\_FORMED.DEFAULT.2012** **INFINITE\_LOOP.GLOBAL** **INFINITE\_LOOP.LOCAL** **INFINITE\_LOOP.MACRO** |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | **48 S, 59 S** | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2023.1 | **CERT\_C-MSC01-a** **CERT\_C-MSC01-b** | All 'if...else-if' constructs shall be terminated with an 'else' clause  The final clause of a switch statement shall be the default clause |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | **474, 744, 787, 9013** | Partially supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2024a | [CERT C: Rec. MSC01-C](https://www.mathworks.com/help/bugfinder/ref/certcrec.msc01c.html) | Checks for missing case for switch condition (rule partially covered) |
| [PVS-Studio](https://wiki.sei.cmu.edu/confluence/display/c/PVS-Studio) | 7.30 | [**V517**](https://pvs-studio.com/en/docs/warnings/v517/)**,** [**V533**](https://pvs-studio.com/en/docs/warnings/v533/)**,** [**V534**](https://pvs-studio.com/en/docs/warnings/v534/)**,** [**V535**](https://pvs-studio.com/en/docs/warnings/v535/)**,** [**V556**](https://pvs-studio.com/en/docs/warnings/v556/)**,** [**V577**](https://pvs-studio.com/en/docs/warnings/v577/)**,** [**V590**](https://pvs-studio.com/en/docs/warnings/v590/)**,** [**V612**](https://pvs-studio.com/en/docs/warnings/v612/)**,** [**V695**](https://pvs-studio.com/en/docs/warnings/v695/)**,** [**V696**](https://pvs-studio.com/en/docs/warnings/v696/)**,** [**V719**](https://pvs-studio.com/en/docs/warnings/v719/)**,** [**V722**](https://pvs-studio.com/en/docs/warnings/v722/)**,** [**V747**](https://pvs-studio.com/en/docs/warnings/v747/)**,** [**V785**](https://pvs-studio.com/en/docs/warnings/v785/)**,** [**V786**](https://pvs-studio.com/en/docs/warnings/v786/) |  |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/c/RuleChecker) | 24.04 | **missing-else**  **switch-default** | Partially checked |
| [SonarQube C/C++ Plugin](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87151949) | 3.11 | [**ElseIfWithoutElse**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-c.html#RSPEC-126)**,** [**SwitchWithoutDefault**](https://www.sonarsource.com/products/codeanalyzers/sonarcfamilyforcpp/rules-c.html#RSPEC-131) |  |

### 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.

Most of the standards defined in this document do not require a special time and place to be addressed, in fact doing so would negate the usefulness of these standards. Rather, these standards should be considered at every stage of the development process. Selecting the right data type is essential when designing and building the program but should not be forgotten about during the later stages like maintenance. Ensuring that data is not lost or misrepresented is vital when verifying and testing the program but is still important when monitoring and detecting possible attacks. Detecting errors when converting a string to a number is important during building and testing, but string manipulation is a common foothold for attacks as well and should be routinely tested during health checks of the system. Sanitizing data is important when testing a program for security risks, but security is a constant battle so monitoring and detecting possible SQL attacks is just as important. Not reading uninitialized memory is important to keep in mind when building a program but it is also important to respond to instances when this does happen as memory deallocation is unpredictable at best when the abort is called. Using the right assertions at the right time and ensuring all exceptions are handled are important both in the testing phase and in the subsequent health checks. Using proper comments is important when building the program, but also important when maintaining it. Finally, striving for logical completeness is important for the system from the planning stage, through the end of the software's lifespan.

### 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 | Low | Unlikely | Medium | Medium | 4 |
| STD-002-CPP | High | Probable | High | Medium | 2 |
| STD-003-CPP | Medium | Unlikely | Medium | Low | 1 |
| STD-004-CPP | High | Likely | Medium | High | 5 |
| STD-005-CPP | High | Probable | Medium | High | 5 |
| STD-006-CPP | Low | Unlikely | High | Low | 1 |
| STD-007-CPP | Low | Probable | Medium | Low | 2 |
| STD-008-CPP | Medium | Unlikely | Medium | Low | 2 |
| STD-009-CPP | Low | Unlikely | Medium | Low | 1 |
| STD-010-CPP | Medium | Probable | 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 | Encrypt all files when being stored and not in use. This means that all data being stored in a physical medium (hard drive, thumb drive, etc.) and virtual medium (database) is encrypted in the event of a breech or theft. |
| Encryption in flight | This is when data is most likely to be misused, so encrypting data as it leaves, enters, and moves around the network is vital. |
| Encryption in use | When data has been sent from a database or storage and has either arrived at the user or is being used by the system, it is important that the data remain encrypted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the process of the system verifying that the user is who the user says they are when the user logs into the system. This ensures that users cannot imitate other users to gain access to areas of the system they should not have access to. |
| Authorization | This is the process of granting authority to a user depending on their level of clearance in the system. This ensures that users are given access only to the areas of the system that they need to perform their tasks in the system and changes made to the system can only be made by those that should be able to make those changes. For example, adding new users and altering the access levels of users should be reserved for individuals like the owner of the system. |
| Accounting | This is the process of tracking and documenting use of the system, particularly the unusual activity in a system. This ensures that in the event of an attack, the source can be identified and rectified as well as knowledge to what was accessed in the attack. |

**\***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 | 06/10/2024 | Initial Document | Dominic Drury |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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