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



Green Pace Secure Development Policy

Benjamin Strickland

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 | Treat all incoming data as untrusted. By validating all data coming in, most vulnerabilities can be prevented. |
| 1. Heed Compiler Warnings | Compilers offer warnings for a reason. Setting the compiler to its highest warning level and handling every warning makes the software more secure. Analysis tools can also be employed to identify more vulnerabilities. |
| 1. Architect and Design for Security Policies | Security should be included in every phase of design so that it is part of the package rather than an afterthought. For example, during the integration phase, it is just as important that the subsystems communicate securely as it is that they communicate at all. |
| 1. Keep It Simple | The more moving parts a thing has, the more places it has that can fail. A more complex system is more difficult to secure. Especially when trying to implement a defense in depth approach, a simpler system is easier to protect. |
| 1. Default Deny | Saying “no” by default keeps the system more secure. The system only says “yes” when the request has been thoroughly inspected and found to be trustworthy. |
| 1. Adhere to the Principle of Least Privilege | A process should only be allowed sufficient privilege to perform its task, and only for as long as that task takes. Then the privilege should be removed. This cuts down on the number of opportunities and the amount of time that a successful attacker has within the system. |
| 1. Sanitize Data Sent to Other Systems | Data being sent to another system, such as a 3rd party or other external program, is susceptible to being injected or otherwise hijacked by an attacker. Controls should be put in place which sanitize the data being sent. It is advised to whitelist allowed characters so the developer can define which ones are acceptable rather than blacklisting since that means the developer has to anticipate every kind of attempt an attacker might try. |
| 1. Practice Defense in Depth | A system is only as secure as its weakest link. However, by implementing multiple layers of links, even a weak link will not cause a failure. Don’t put too much trust in a single defensive layer. Rather, build multiple, overlapping layers of defense around the process to better protect it. |
| 1. Use Effective Quality Assurance Techniques | Testing should be thorough and ongoing. Further, when possible, testing should be done by a 3rd party. This will reveal more vulnerabilities because external entities will not be operating under the same assumptions as an internal developer. |
| 1. Adopt a Secure Coding Standard | These principles are statements of commitment. But standards are mandatory rules that ensure the intentions of the principles are being met. By adopting standards, we are creating rules that offer specific, workable security solutions. |

## 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** | **Do not cast to an out-of-range enumeration value** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | To avoid operating on unspecified values, the arithmetic value being cast must be within the range of values the enumeration can represent. When dynamically checking for out-of-range values, checking must be performed before the cast expression. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example is casting to an integer data type before checking whether it is in bounds. This could cause a boundary-related exception to occur. |
| enum myType{  Zero,  One,  Two};  void foo(int myInt){  myType myVar = static\_cast<myType>(myInt);  if(myVar < Zero || myVar > Two){  // Boundary check  }  } |

| **Compliant Code** |
| --- |
| The compliant code example checks the boundaries of the integer parameter to ensure it falls within the boundaries of the enumerator value. |
| enum myType{  Zero,  One,  Two};  void foo(int myInt){  if(myInt < Zero || myInt > Two){  //Handle out of range error  }  myType myVar = static\_cast<myType>(myInt); |

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

|  |
| --- |
| **Principles(s):**  **1. Validate Input Data:** We should treat all incoming data as untrusted. By installing boundary checks prior to casting, we can prevent overflow vulnerabilities.  **7. Sanitize Data Sent to Other Systems:** The incoming data is being sanitized to prevent it from causing corruption as could be the case by casting to an out-of-range enumeration value. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 | <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Axivion+Bauhaus+Suite> |
| Helix QAC | 2021.2 | C++3013 | <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC> |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-INT50-a | <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft> |
| PVS-Studio | 7.13 | V1016 | <https://wiki.sei.cmu.edu/confluence/display/cplusplus/PVS-Studio> |

### Coding Standard 2

| **Coding Standard** | **Label** | **Ensure that unsigned integer operations do not wrap** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Certain operations involving unsigned integers have the possibility of wrapping around if the result of the operation creates a value that cannot be contained within the limits of an integer value. |

| **Noncompliant Code** |
| --- |
| The noncompliant code example shows a function that simply adds two unsigned integers together. There is the possibility that the value assigned to sum will be larger than what an unsigned integer can hold, causing it to wrap. |
| void myFunc(unsigned int a, unsigned int b){  unsigned int sum = a + b;  } |

| **Compliant Code** |
| --- |
| The compliant code example shows a function that checks to ensure that the sum of the two unsigned integer values is less than the maximum allowed value. If not, the error is caught and handled. If so, the operation continues. |
| void myFunc(unsigned int a, unsigned int b){  if(UINT\_MAX – a < b){  // Handle wrapping error  }else{  unsigned int sum = a + b;  }  } |

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

|  |
| --- |
| **Principles(s):**  **1. Validate Input Data:** The parameters to the function may come from an untrusted source. We should verify the parameters are within the limits of the incoming variable’s data type as well as the resulting variable’s data type since we are adding them together.  **7. Sanitize Data Sent to Other Systems:** This function may add the two parameters together and then send the result elsewhere. We should do our due diligence to ensure that the returned variable is not susceptible to an overflow which would then be sent on to another system.  **8. Practice Defense in Depth:** This is an added layer of security that has little impact on performance. Defense in depth is about covering as many potential vulnerabilities as we can. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | Integer-overflow | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-INT30 | Implemented |
| CodeSonar | 6.0p0 | ALLOC.SIZE.IOFLOW | Integer overflow of allocation size |
| TrustInSoft Analyzer | 1.38 | Unsigned overflow | Exhaustively verified. |

### Coding Standard 3

| **Coding Standard** | **Label** | **Guarantee that storage for strings has sufficient space for character data and the null terminator** |
| --- | --- | --- |
| **String Correctness** | STD-003-CLG | Trying to install data into a string that does not sufficient space for all the characters and the null terminator can result in a buffer overflow. |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, the input is unbounded, and could lead to a buffer overflow. |
| void foo(){  char myString[4];  std::cin >> myString;  } |

| **Compliant Code** |
| --- |
| In this compliant code example, cin has a set width, which will truncate any extra characters that do not fit. |
| void foo(){  char myString[4];  std::cin.width(5);  std::cin >> myString;  } |

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

|  |
| --- |
| **Principles(s):**  **3. Architect and Design for Security Policies:** Security starts at the beginning of the process and is included in every step. By thinking ahead with a security mindset, we can generate code is that has fewer vulnerabilities. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | BUFFER\_SIZE | Supported  Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR31 | Detects calls to unsafe string function that may cause buffer overflow Detects potential buffer overruns, including those caused by unsafe usage of fscanf() |
| CodeSonar | 6.0p0 | **LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.\*** | Buffer overrun Type overrun No space for null terminator A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Coverity | 2017.07 | **STRING\_OVERFLOW**  **BUFFER\_SIZE**  **OVERRUN**  **STRING\_SIZE** | Fully implemented |

### Coding Standard 4

| **Coding Standard** | **Label** | **Prevent SQL Injection** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | When raw user input is used to build a string, which becomes a SQL query, there is the possibility that an attacker can supply purposefully malicious input to obtain sensitive data, such as passwords, from a database. |

| **Noncompliant Code** |
| --- |
| This noncompliant code example shows a situation where a user could input malicious text into the username and/or password fields to cause the server to respond with data from the database without being authenticated. |
| void authenticate(String username, String password){  String login = “SELECT \* FROM db WHERE username = ‘” + username + “’ AND password = ‘” + password + “’”;  } |

| **Compliant Code** |
| --- |
| This compliant code example shows how to parameterize the inputs so that they can be better validated and less likely to be susceptible to an injection attack. |
| void authenticate(String username, String password){  String login = “SELECT \* FROM db WHERE username = %s AND password = %s”, username, password;  } |

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

|  |
| --- |
| **Principles(s):**  **6. Adhere to the Principle of Least Privilege:** Only the users with proper authorization should be allowed to perform certain tasks. If the malicious user lacks authorization to perform the task, then their attempt is thwarted.  **7. Sanitize Data Sent to Other Systems:** This example shows a query being built from strings which is then sent to the database to retrieve data. We need to do as much as we can to limit the amount of input that the user can inject into a system, and what little input goes in is sanitized.  **8. Practice Defense in Depth:** This is an added layer of security that prevents untrusted data from reaching sensitive areas of the system. The more layers we can add, the deeper our depth of defense. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors.  <https://wiki.sei.cmu.edu/confluence/display/java/The+Checker+Framework> |
| Coverity | 7.5 | **SQLI FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_** **FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented  <https://wiki.sei.cmu.edu/confluence/display/java/Coverity> |
| Findbugs | 1.0 | **SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE** | Implemented  <https://wiki.sei.cmu.edu/confluence/display/java/Findbugs> |
| Fortify | 1.0 | **HTTP\_Response\_Splitting** **SQL\_Injection\_\_Persistence** **SQL\_Injection** | Implemented  <https://wiki.sei.cmu.edu/confluence/display/java/Fortify> |

### Coding Standard 5

| **Coding Standard** | **Label** | **Do not Access Freed Memory** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Accessing a deallocated pointer’s memory location is undefined behavior. Always make sure that a pointer is not destroyed until it is no longer needed, and do not reference it after it has been destroyed. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a new pointer p being declared. Then the memory location it points to is assigned the value of 1. Next, the pointer is deleted. Finally, the value of 2 is assigned to the memory location that was pointed to by p, which is undefined behavior. |
| int \*p;  \*p = 1;  delete p;  \*p = 2; |

| **Compliant Code** |
| --- |
| The following compliant code example shows that the assignments to the memory location pointed to by p take place before p is deallocated. Once p has been deleted, no more references to p should occur. |
| int \*p;  \*p = 1;  \*p = 2;  delete p; |

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

|  |
| --- |
| **Principles(s):**  **10. Adopt a Secure Coding Standard:** It would be an easy mistake to try to reuse a deleted pointer, or to delete a pointer before it is done being used. By instituting these standards, we can prevent this type of undefined behavior. It gives all of the developers on the project a set of rules that help add layers of defense. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory  <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428> |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations  <https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite> |
| CodeSonar | 6.1p0 | ALLOC.UAF | Use after free  <https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar> |
| PC-lint Plus | 1.4 | 449, 2434 | Fully supported  <https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus> |

### Coding Standard 6

| **Coding Standard** | **Label** | **Do not use assertions as part of the normal flow of code** |
| --- | --- | --- |
| **Assertions** | STD-006-CLG | Assertions are useful for testing purposes, but they should not be used as part of the process of the program’s functionality. Using assertions in this manner means the program will not function properly when assertions are later turned off. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows that assert is being used to remove all the NULLs from the names arraylist. However, when assertions are disabled, this function no longer works correctly. |
| ArrayList<String> names;  void noNullNames(){  assert names.remove(null);  } |

| **Compliant Code** |
| --- |
| The following compliant code example shows how all NULL names have been removed from the list, and the assertion only tests the Boolean variable, which will not cause side effects when assertions are turned off. |
| ArrayList<String> names;  void noNullNames(){  Boolean noNulls = names.remove(null);  assert noNulls;  } |

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

|  |
| --- |
| **Principles(s):**  **4. Keep it Simple:** While combining tasks makes for fewer lines of code, setting up each task on its own line improves readability but also helps to prevent unintended side effects such as the one shown in the noncompliant code example.  **9. Use Effective Quality Assurance Techniques:** Using assertions is a good idea for testing, but we need to be careful that no production code is involved with the assertions since they will be turned off at some point. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | JAVA.STRUCT.SE.ASSERT | Assertion Contains Side Effects (Java)  <https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar> |
| PVS-Studio | 7.13 | V6055 | <https://pvs-studio.com/en/docs/warnings/v6055/> |
| SonarQube | 6.7 | S3346 | Expressions used in “assert” should not produce side effects  <https://rules.sonarsource.com/java/RSPEC-3346> |

### Coding Standard 7

| **Coding Standard** | **Label** | **Handle all Exceptions** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | A thrown exception transfers control to the nearest exception handler with a matching type. If no handler can be found, the function std::terminate() is called, which calls std::abort(), which abnormally terminates the process, leaving the system in an indetermined state. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows how neither the main function nor the foo function can handle the exception thrown by the function throws\_an\_exception. This results in std::terminate() being called, which aborts the process. |
| void throws\_an\_exception(){}  void foo(){  throws\_an\_exception();  }  int main(){  foo();  } |

| **Compliant Code** |
| --- |
| The following compliant code example shows how the main function calls the foo function from within a try-catch clause, which will capture the exception thrown by throws\_an\_exception, allowing the program to fail gracefully without aborting. |
| void throws\_an\_exception(){}  void foo(){  throws\_an\_exception();  }  int main(){  try{  foo();  } catch(…){  //Handle error  }  } |

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

|  |
| --- |
| **Principles(s):**  **3. Architect and Design for Security Policies:** Unhandled exceptions decreases the user’s trust in the software. More importantly, unhandled exceptions leave vulnerabilities that can be exploited by bad actors. Further, catch-all errors should not be hidden by failing to catch and properly report an error that was caught.  **8. Practice Defense in Depth:** Making sure all exceptions have a handler allows the software to fail gracefully and mitigates vulnerabilities. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | main-function-catch-all  early-catch-all | Partially checked  <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724> |
| Parasoft C/C++test | 2021.1 | 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  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft> |
| Polyspace Bug Finder | R2021a | CERT C++:ERR51-CPP | Checks for unhandled exceptions (rule partially covered)  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder> |
| RuleChecker | 20.10 | main-function-catch-all  early-catch-all | Partially checked  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker> |

### Coding Standard 8

| **Coding Standard** | **Label** | **Ensure that container iterators and indices are within range** |
| --- | --- | --- |
| **Containers** | STD-008-CPP | Sending the iterator and index value of a container as a parameter leaves a function open to untrusted data. Always check the boundaries of such data. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code shows a function that will access the element at the position parameter. However, position is an int value which can be very large and can also be a negative value. Either of which can attempt to access an element that is out of range. |
| void accessElement(int numberList[], int listSize, int position){  int element = numberList[position];  return element;  } |

| **Compliant Code** |
| --- |
| The following compliant code shows that the position parameter (and the size of the list) has been sent as an unsigned int, which cannot be negative. |
| void accessElement(int numberList[], unsigned int listSize, unsigned int position){  if(position < listSize){  int element = numberList[position];  return element;  }  } |

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

|  |
| --- |
| **Principles(s):**  **1. Validate Input Data:** Having the iterator and/or index value passed as a parameter means those variables are untrusted. We should validate their accuracy to ensure that the function is not accessing elements out of bounds.  **7. Sanitize Data Sent to Other Systems:** The element gathered in the example above is being returned to the calling function. We should ensure that the data is accurate by controlling how the container is accessed. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | overflow\_upon\_dereference | <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724> |
| CodeSonar | 6.1p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TO  LANG.MEM.TU  LANG.MEM.TBA  LANG.STRUCT.PBB  LANG.STRUCT.PPE | Buffer overrun  Buffer underrun  Type overrun  Type underrun  Tainted buffer access  Pointer before beginning of object  Pointer past end of object |
| Helix QAC | 2021.2 | C++2891, C++3139, C++3140 | <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC> |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft> |

### Coding Standard 9

| **Coding Standard** | **Label** | **Always properly close file access when done. Do not allow a file to remain open after program execution.** |
| --- | --- | --- |
| **File Handling** | STD-009-CPP | Files must be opened prior to working with their contents. Likewise, files must be closed when done. Files left open are in an undefined state, leading to vulnerabilities. |

| **Noncompliant Code** |
| --- |
| In the following noncompliant code example, a file is opened, and some work is performed on its contents. Then the function returns normally but the file is left open, which could create a vulnerability. |
| void readFile(std::string filename){  std::fstream file(filename);  if(!file.is\_open()){  //handle error  }  //some work on the file contents  return;  } |

| **Compliant Code** |
| --- |
| The following compliant code example shows the file being opened and some work being done on its contents. When the work is complete, the file is properly closed, preventing any vulnerabilities due to it being left open. |
| void readFile(std::string filename){  std::fstream file(filename);  if(!file.is\_open()){  //handle error  }  //some work on the file contents  file.close();  return;  } |

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

|  |
| --- |
| **Principles(s):**  **10. Adopt a Secure Coding Standard:** This standard is not a guideline, but a rule that must be followed with every file access. Leaving files open after the program terminates leaves it in a vulnerable state, so we need to be sure that we close every file. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | ALLOC.LEAK | Leak  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar> |
| Helix QAC | 2021.2 | C++4786, C++4787, C++4788 | <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Helix+QAC> |
| Klocwork | 2021.1 | RH.LEAK | <https://docs.roguewave.com/en/klocwork/current/> |
| Parasoft C/C++test | 2021.1 | CERT\_CPP-FIO51-a | Ensure resources are freed  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft> |

### Coding Standard 10

| **Coding Standard** | **Label** | **Do not reference uninitialized variables** |
| --- | --- | --- |
| **Variable Use** | STD-010-CPP | Reading from an uninitialized variable is bad practice because the data stored there could be anything, or nothing, both of which are undefined behavior. |

| **Noncompliant Code** |
| --- |
| The following noncompliant code example shows a function that names a variable but does not initialize it. Then the contents of that variable are output to the console. What is output to the console is unknown, and it is just as likely that it will cause an error if there is a data type mismatch or there is nothing at that memory location. |
| void foo(){  int a;  std::cout << a;  } |

| **Compliant Code** |
| --- |
| The following compliant code example properly initializes the variable, ensuring that something valid is in that memory location to then output to the console. |
| void foo(){  int a = 0;  std::cout << a;  } |

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

|  |
| --- |
| **Principles(s):**  **10. Adopt a Secure Coding Standard:** Standards such as this one are not guidelines, but rules that must be followed. Failure to initialize a variable prior to using it can result in undefined behavior since we have no way of knowing what data is stored at that location. This leaves the program vulnerable to exceptions. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | uninitialized-read | Partially checked  <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724> |
| Clang | 3.9 | -Wuninitialized  Clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory.  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/Clang> |
| CodeSonar | 6.1p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local uninitialized variable  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar> |
| LDRA tool suite | 9.7.1 | 53 D, 69 D, 631 S, 652 S | Partially implemented  <https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA> |

## Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



# Project One

There are seven steps outlined below that align with the elements you will be graded on in the accompanying rubric. When you complete these steps, you will have finished the security policy.

## Revise the C/C++ Standards

You completed one of these tables for each of your standards in the Module Three milestone. In Project One, add revisions to improve the explanation and examples as needed. Add rows to accommodate additional examples of compliant and noncompliant code. Coding standards begin on the security policy.

## Risk Assessment

Complete this section on the coding standards tables. Enter high, medium, or low for each of the headers, then rate it overall using a scale from 1 to 5, 5 being the greatest threat. You will address each of the seven policy standards. Fill in the columns of severity, likelihood, remediation cost, priority, and level using the values provided in the appendix.

## Automated Detection

Complete this section of each table on the coding standards to show the tools that may be used to detect issues. Provide the tool name, version, checker, and description. List one or more tools that can automatically detect this issue and its version number, name of the rule or check (preferably with link), and any relevant comments or description—if any. This table ties to a specific C++ coding standard.

## Automation

Provide a written explanation using the image provided.



Automation will be used for the enforcement of and compliance to the standards defined in this policy. Green Pace already has a well-established DevOps process and infrastructure. Define guidance on where and how to modify the existing DevOps process to automate enforcement of the standards in this policy. Use the DevSecOps diagram and provide an explanation using that diagram as context.

The first place that automation should be included is the Design stage. Test-driven designs can benefit from automation because the test can run automatically and inform the design team where vulnerabilities might lie. This stage can also benefit from OWASP and other security testing services. The next stage is Build and it can also benefit from automation. At this stage we can implement static testing tools to identify vulnerabilities in open source and 3rd party repositories.

The verify and test stage can also benefit from automation. This stage includes vulnerability scanning and functional testing which can utilize automated tools to detect vulnerabilities. The following stage, Transition and health check, can likewise benefit from automation. Once the process for transition has been finalized, we can build tools to make this process automatic. If this automated process can also collect logs on issues encountered then we will more easily be able to automate the next step, Monitor and detect.

The Monitor and detect stage can benefit from automation by collecting logs on errors and build analytics to report on new and ongoing issues. This automation can also be utilized to feed into the next stage, where we respond to the issues identified. The response can likewise be automated to some extent. The best places for this kind of automation are blocking attacks and turning off services that come under attack.

Finally, the Maintain and stabilize stage can benefit from automation by having tools in place to return the system to a secure baseline in the event of an attack or compromise. These automated tools can also collect and format all the data collected and feed it into the first stage of pre-production where we Assess and plan. At this point, we can better assess the threat landscape and be better prepared for the following Design phase. With each iteration through this loop, our tools become more accurate, more secure, and more efficient.

## 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 | Medium | Unlikely | Medium | P4 | L3 |
| STD-002-CPP | High | Likely | High | P9 | L2 |
| STD-003-CLG | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Probable | Medium | P12 | L1 |
| STD-005-CPP | High | Likely | Medium | P18 | L1 |
| STD-006-CLG | Low | Unlikely | Low | P3 | L3 |
| STD-007-CPP | Low | Probable | Medium | P4 | L3 |
| STD-008-CPP | High | Likely | High | P9 | L2 |
| STD-009-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-010-CPP | High | Probable | Medium | P12 | L1 |

## Create Policies for Encryption and Triple A

Include all three types of encryption (in flight, at rest, and in use) and each of the three elements of the Triple-A framework using the tables provided***.***

* 1. Explain each type of encryption, how it is used, and why and when the policy applies.
  2. Explain each type of Triple-A framework strategy, how it is used, and why and when the policy applies.

Write policies for each and explain what it is, how it should be applied in practice, and why it should be used.

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Data that is stored and not currently in use needs to be protected. The best practice is to encrypt data at rest so that in the event of unauthorized access, the data is still secure. This is because the data is in a form that cannot be read without the key. This also follows along with the principles 5. Default Deny and 6. Adhere to the Principle of Least Privilege. |
| Encryption at flight | Data that has been requested from a server by a client needs to be protected while it is in transit. Since it is possible to perform man-in-the-middle attacks, and our reliance on the cloud is increasing, it is imperative that data is encrypted while it is being transferred between systems. This is supported by principles 7. Sanitize Data Sent to Other Systems and 8. Practice Defense in Depth. |
| Encryption in use | Data that is in use by a program still needs to be protected. There is a concern about the decrease in performance this can cause, but there are ways to counteract this, such as selective encryption of database fields, rows, and columns versus encrypting all data regardless of sensitivity. Data in use can be further protected by enforcing access rules such as role-based access control and multi-factor authentication, which can prevent unauthorized access to the data beyond encryption. This is supported by principles 5. Default Deny, 6. Adhere to the Principle of Least Privilege, and 8. Practice Defense in Depth. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the process of verifying that the user is who they claim to be. This is primarily done by ensuring they know something that nobody else does, such as a password. However, a user can also be authenticated by what they have, such as a keycard. Since passwords can be guessed and keycards can be stolen, however, it is a good practice to use multi-factor authentication, which would require both the password and keycard in conjunction, increasing security against unauthenticated access. |
| Authorization | Authorization is the process of ensuring that an authenticated user is allowed to do the task they are trying to perform. One approach to this is role-based access control, where users are grouped by the roles they fulfill, and they are only given access to those actions necessary to complete their assigned tasks. According to the Principle of Least Privilege, a user or role should only be given enough privilege to complete their specific tasks, and no more. This prevents users from accidentally performing tasks they should not. It also limits the amount of access that users have to sensitive data, reducing the attack surface, adding another layer to the defense in depth approach. |
| Accounting | Even with multi-factor authentication and the Principle of Least Privilege, it is still possible for a bad actor to gain access to sensitive areas of the system. This is why accounting is so important. It is the process of monitoring traffic and keeping logs of all activity in the system. Without proper accounting processes in place, we might not know that an intrusion took place, and much less be able to do something about it. This is the importance of accounting. We need to know who gained access to which parts of the system and at what times in order to better understand the methods the intruder used to gain access. With all that information in hand, we will be able to design a defense against that type of intrusion, thereby adding more layers to our defense in depth strategy. |

**\***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 | 07/16/2021 | Principles and Standards | Ben Strickland |  |
| 1.2 | 08/04/2021 | Beginning Risk Assessments | Ben Strickland |  |
| 1.3 | 08/05/2021 | Cont’d Risk Assessments | Ben Strickland |  |
| 1.4 | 08/06/2021 | Completed Encryption and Triple-A explanations | Ben Strickland |  |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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