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



Green Pace Secure Development Policy

Janera Dobson

Southern New Hampshire University

CS-405-H4736

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 | To validate input data is the process of ensuring the data a user inputs falls within the intended program. In order to execute this behavior, the data inputs must obey the type and range of requirements to eliminate software vulnerabilities. Therefore, the data will go through a number of tests to prevent improperly formed data from entering an information system. |
| 1. Heed Compiler Warnings | When compiling code, you should use the highest warning available for the compiler and eliminate warnings by modifying the code. Reason being is because the compiler was designed with built-in warnings for a reason. Therefore, it’s important to eliminate all warnings by correcting the code, not by changing the compiler to a lower warning level and not ignore them. |
| 1. Architect and Design for Security Policies | Architect and design for security policies implement and enforce security policies. They look at how information security controls and safeguards are implemented in IT systems in order to protect the confidentiality, integrity, and availability of the data that are used, processed, and stored in those systems. |
| 1. Keep It Simple | Keeping it simple is the best way to not complicate your program design. Hence, complex designs increase the chances of errors to occur. Therefore, keeping a design principle simple guarantees the greatest levels of user acceptance and interaction. |
| 1. Default Deny | Default deny allows permission based on decisions rather than exclusion. Basically, it means unless you specifically allow something, you deny it. It’s a networks version of whitelisting and most cases in the perimeter of your firewall; it defines the ports and protocols you allow and turns everything else off. |
| 1. Adhere to the Principle of Least Privilege | Adhering to the principle of least privilege works by allowing only enough access to perform the required job. Doing so, reduces the risk of attackers gaining access to critical systems or sensitive data by compromising a low-level user account, device, or application. Therefore, every process should execute with the least set of privileges necessary to complete the job. |
| 1. Sanitize Data Sent to Other Systems | Sanitize data dent to other systems is the process of deliberately, permanently, and irreversibly removing or destroying the data stored on a memory device to make it unrecoverable. Sanitizing data modifies the input to ensure that it is valid especially information sent with doubled single quotes. This would normally occur when you combine these two techniques to provide in-depth defense to your application. |
| 1. Practice Defense in Depth | Practice defense in depth manages risk with multiple defensive strategies, so if one layer was penetrated then there’s another layer to prevent a security flaw from being exploited. This security mechanism protects the confidentiality, integrity, and availability of the network and data within. |
| 1. Use Effective Quality Assurance Techniques | Use effective quality assurance techniques is a good technique for identifying and eliminating vulnerabilities. The planned and systematic activities implemented in a quality system so that quality requirements for a product or service will be fulfilled. Quality assurance methods, therefore, tend to be process driven and are primarily focused on the development of the product or the delivery of a service. |
| 1. Adopt a Secure Coding Standard | It is important to adopt a secure coding standard because they are rules and guidelines used to prevent security vulnerabilities. When used effectively, these security standards prevent, detect, and eliminate errors that could compromise software security for whatever your choice of development language and platform may be. |

## C/C++ Ten Coding Standards

Complete the coding standards portion of the template according to the Module Three milestone requirements. In Project One, follow the instructions to add a layer of security to the existing coding standards. Please start each standard on a new page, as they may take up more than one page. The first seven coding standards are labeled by category. The last three are blank so you may choose three additional standards. Be sure to label them by category and give them a sequential number for that category. Add compliant and noncompliant sections as needed to each coding standard.

### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-111-C | Implement abstract data types using opaque types |

| **Noncompliant Code** |
| --- |
| * The string type and functions that operate on this function are defined in the string\_m.h header. * string\_m.h is fully visible to the user, allowing them to manipulate fields within the structure, violate coding principles and increases the chances of flaws and the program becoming vulnerable. |
| struct string\_mx {  **size\_t** size;  **size\_t** maxsize;    unsigned **char** strtype;  **char** \*cstr;  };    typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/ |

| **Compliant Code** |
| --- |
| * Struct string\_mx is fully defined, but not visible to users of the abstract data. * string\_mx type is now private which will hide the data type from the user. There’s an external string\_m.h file and string\_mx type which is an instance of string\_mx, declaring it as an incomplete type. |
| struct string\_mx {  **size\_t** size;  **size\_t** maxsize;    unsigned **char** strtype;  **char** \*cstr;  };  struct string\_mx;  typedef struct string\_mx string\_mx;    /\* Function declarations \*/  extern errno\_t strcpy\_m(string\_mx \*s1, const string\_mx \*s2);  extern errno\_t strcat\_m(string\_mx \*s1, const string\_mx \*s2);  /\* ... \*/ |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Use Effective Quality Assurance Techniques, and Adopt a Secure Coding Standard.  Programmers are consequently more likely to directly manipulate the fields within the structure, violating the software engineering principles of information hiding and data encapsulation and increasing the probability of developing incorrect or nonportable code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 6.9.0 | CertC- DCL12 |  |
| LDRA tool suite | 9.7.1 | 104 D | Partially implemented |
| Polyspace Bug Finder | R2020a | CERT C: Rec. DCL12-C | Checks for structure or union object implementation visible in file where pointer to this object is not dereferenced (rule partially covered) |
| Parasoft C/C++test | 2020.2 | CERT\_C-DCL12-a | If a pointer to a structure or union is never dereferenced within a translation unit, then the implementation of the object should be hidden |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-222-C | [Ensure that integer conversions do not result in lost or misinterpreted data](https://wiki.sei.cmu.edu/confluence/display/c/INT31-C.+Ensure+that+integer+conversions+do+not+result+in+lost+or+misinterpreted+data) |

| **Noncompliant Code** |
| --- |
| This code example results in a truncation and sign errors during implementation due to unsigned type to signed type conversion. |
| #include <limits.h>    void func(void) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;    sc = (**signed** **char**)u\_a; /\* Cast eliminates warning \*/    /\* ... \*/  } |

| **Compliant Code** |
| --- |
| The solution would be to use an unsigned long int type to convert the value to a signed char type. |
| #include <limits.h>    void func(void) {    unsigned **long** **int** u\_a = ULONG\_MAX;  **signed** **char** sc;    if (u\_a <= SCHAR\_MAX) {      sc = (**signed** **char**)u\_a;  /\* Cast eliminates warning \*/    } else {      /\* Handle error \*/    }  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Use Effective Quality Assurance Techniques, Heed Compiler Warnings, and Adopt a Secure Coding Standard.  Integer conversions, both implicit and explicit (using a cast), must be guaranteed not to result in lost or misinterpreted data. Consequently, it is not always possible to represent all possible values of an unsigned short int as an int. Integer truncation errors can lead to buffer overflows and the execution of arbitrary code by an attacker. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | LANG.CAST.PC.AV LANG.CAST.PC.CONST2PTR LANG.CAST.PC.INT  LANG.CAST.COERCE LANG.CAST.VALUE  ALLOC.SIZE.TRUNC MISC.MEM.SIZE.TRUNC  LANG.MEM.TBA | Cast: arithmetic type/void pointer Conversion: integer constant to pointer Conversion: pointer/integer  Coercion alters value Cast alters value  Truncation of allocation size Truncation of size  Tainted buffer access |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity)\* | 2017.07 | NEGATIVE\_RETURNS  REVERSE\_NEGATIVE  MISRA\_CAST | Can find array accesses, loop bounds, and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior  Can find instances where a negativity check occurs after the negative value has been used for something else  Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted |
| [Cppcheck](https://wiki.sei.cmu.edu/confluence/display/c/Cppcheck) | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | CERT\_C-INT31-a CERT\_C-INT31-b CERT\_C-INT31-c CERT\_C-INT31-d CERT\_C-INT31-e CERT\_C-INT31-f CERT\_C-INT31-g CERT\_C-INT31-h CERT\_C-INT31-i CERT\_C-INT31-j CERT\_C-INT31-k CERT\_C-INT31-l CERT\_C-INT31-m CERT\_C-INT31-n CERT\_C-INT31-o | 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 integer overflows |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-333-C | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| This example converts a string stored in buff to a signed int value using atoi() function which lack the mechanism for reporting errors for invalid values. |
| #include <stdlib.h>    void func(const **char** \*buff) {  **int** si;      if (buff) {      si = **atoi**(buff);    } else {      /\* Handle error \*/    }  } |

| **Compliant Code** |
| --- |
| This solution uses strtol() in order to convert a string to an int which would ensure the result value after implemented to be in the range of an int. |
| #include <errno.h>  #include <limits.h>  #include <stdlib.h>  #include <stdio.h>    void func(const **char** \*buff) {  **char** \*end;  **int** si;    **errno** = 0;      const **long** sl = **strtol**(buff, &end, 10);      if (end == buff) {  **fprintf**(stderr, "%s: not a decimal number\n", buff);    } else if ('\0' != \*end) {  **fprintf**(stderr, "%s: extra characters at end of input: %s\n", buff, end);    } else if ((LONG\_MIN == sl || LONG\_MAX == sl) && ERANGE == **errno**) {  **fprintf**(stderr, "%s out of range of type long\n", buff);    } else if (sl > INT\_MAX) {  **fprintf**(stderr, "%ld greater than INT\_MAX\n", sl);    } else if (sl < INT\_MIN) {  **fprintf**(stderr, "%ld less than INT\_MIN\n", sl);    } else {      si = (**int**)sl;        /\* Process si \*/    }  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Heed Compiler Warnings, Keep It Simple, Use Effective Quality Assurance Techniques, and Adopt a Secure Coding Standard.  The process of parsing an integer or floating-point number from a string can produce many errors. The string might not contain a number. It might contain a number of the correct type that is out of range (such as an integer that is larger than INT\_MAX). These error conditions must be detected and addressed when a string-to-number conversion is performed using a C Standard Library function. Use one of the C Standard Library strto\*() functions to parse an integer or floating-point number from a string. These functions provide more robust error handling than alternative solutions. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [CERT C: Rule ERR34-C](https://www.mathworks.com/help/bugfinder/ref/certcruleerr34c.html) | Checks for unsafe conversion from string to numeric value (rule fully covered) |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | cert-err34-c | Checked by clang-tidy |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 44 S | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | CERT\_C-ERR34-a | The library functions atof, atoi and atol from library stdlib.h shall not be used |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-444-C | Sanitize data passed to complex subsystems |

| **Noncompliant Code** |
| --- |
| This example inputs an email address to a buffer and then uses a string as an argument in a call to system() which causes problems because the user can input the wrong string as an email address. |
| **sprintf**(buffer, "/bin/mail %s < /tmp/email", addr);  **system**(buffer); |

| **Compliant Code** |
| --- |
| The solution would be ensuring all data is valid and accepted and if deemed vulnerable be sanitized. In doing so, based on the tcp\_wrappers package, this whitelisting approach sanitizes the defined list of acceptable characters and eliminates any characters that is not. |
| 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):** Sanitize Data Sent to Other Systems, Architect and Design for Security Policies, Use Effective Quality Assurance Techniques, and Adopt a Secure Coding Standard.  String data passed to complex subsystems may contain special characters that can trigger commands or actions, resulting in a software [vulnerability](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability). As a result, it is necessary to [sanitize](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-sanitize) all string data passed to complex subsystems so that the resulting string is innocuous in the context in which it will be interpreted. It is necessary to ensure that all valid data is accepted, while potentially dangerous data is rejected or [sanitized](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-sanitize). Doing so can be difficult when valid characters or sequences of characters also have special meaning to the subsystem and may involve [validating](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-validation) the data against a grammar. In cases where there is no overlap, whitelisting can be used to eliminate dangerous characters from the data. As a result, the programmer can be less concerned about which characters an attacker may try in an attempt to bypass security checks. Failure to [sanitize](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-sanitize) data passed to a complex subsystem can lead to an injection attack, data integrity issues, and a loss of sensitive data. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | 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 |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | CERT\_C-STR02-a CERT\_C-STR02-b CERT\_C-STR02-c | Protect against command injection Protect against file name injection Protect against SQL injection |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [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** |
| --- | --- | --- |
| **Memory Protection** | STD-555-C | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| This example shows the incorrect way of freeing the memory associated with a linked list. It was done by p being freed before p ->next was executed, therefore, p-> reads memory that has aready been freed. |
| #include <stdlib.h>    struct node {  **int** value;    struct node \*next;  };    void free\_list(struct node \*head) {    for (struct node \*p = head; p != NULL; p = p->next) {  **free**(p);    }  } |

| **Compliant Code** |
| --- |
| This solution shows the correct was to free up memory by storing a reference to p->next in q before freeing p. |
| #include <stdlib.h>    struct node {  **int** value;    struct node \*next;  };    void free\_list(struct node \*head) {    struct node \*q;    for (struct node \*p = head; p != NULL; p = q) {      q = p->next;  **free**(p);    }  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Sanitize Data Sent to Other Systems, Architect and Design for Security Policies, Use Effective Quality Assurance Techniques, and Adopt a Secure Coding Standard.  Reading memory that has already been freed can lead to abnormal program termination and denial-of-service attacks. Writing memory that has already been freed can additionally lead to the execution of arbitrary code with the permissions of the vulnerable process. Programmers should be wary when freeing memory in a loop or conditional statement; if coded incorrectly, these constructs can lead to double-free vulnerabilities. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=87152428) | 20.10 | dangling\_pointer\_use | Supported  Astrée reports all accesses to freed allocated memory. |
| [Axivion Bauhaus Suite](https://wiki.sei.cmu.edu/confluence/display/c/Axivion+Bauhaus+Suite) | 6.9.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | ALLOC.UAF | Use after free |
| [Coverity](https://wiki.sei.cmu.edu/confluence/display/c/Coverity) | 2017.07 | USE\_AFTER\_FREE | Can detect the specific instances where memory is deallocated more than once or read/written to the target of a freed pointer |

### 

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-666-C | Use a static assertion to test the value of a constant expression |

| **Noncompliant Code** |
| --- |
| This example uses the assert() method to assert a property concerning a memory-mapped structure to behave correctly. |
| #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** |
| --- |
| Static assertions allow incorrect assumptions to be diagnosed at compile time with no cost to runtime or time is incurred. Therefore, an assertion can be used at file or block scope, and failure results in a meaningful and informative diagnostic error message. |
| |  | | --- | | #include <assert.h>    struct timer {    unsigned **char** MODE;    unsigned **int** DATA;    unsigned **int** COUNT;  };    static\_assert(sizeof(struct timer) == sizeof(unsigned **char**) + sizeof(unsigned **int**) +  sizeof(unsigned **int**), "Structure must not have any padding"); | |  | |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Sanitize Data Sent to Other Systems, Architect and Design for Security Policies, Use Effective Quality Assurance Techniques, and Adopt a Secure Coding Standard.  Assertions are a valuable diagnostic tool for finding and eliminating software defects that may result in [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability). The runtime assert() macro has some limitations, however, in that it incurs a runtime overhead and because it calls abort(). Consequently, the runtime assert() macro is useful only for identifying incorrect assumptions and not for runtime error checking. As a result, runtime assertions are generally unsuitable for server programs or embedded systems. Static assertion is a valuable diagnostic tool for finding and eliminating software defects that may result in [vulnerabilities](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-vulnerability) at compile time. The absence of static assertions, however, does not mean that code is incorrect. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Clang](https://wiki.sei.cmu.edu/confluence/display/c/Clang) | 3.9 | misc-static-assert | Checked by clang-tidy |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/c/CodeSonar) | 6.0p0 | (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-assertinstead; this assumes ROSE can recognize macro invocation |
| [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** |
| --- | --- | --- |
| **Exceptions** | STD-777-CPP | Honor exception specifications |

| **Noncompliant Code** |
| --- |
| This example shows a function declared as nonthrowing, but may throw an exception when memory cannot be allocated by the std::vector::resize() function. |
| #include <cstddef>  #include <vector>    void f(std::vector<**int**> &v, **size\_t** s) noexcept(true) {    v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| This solution removes the noexcept-specification and signifies that the function allows all exceptions. |
| #include <cstddef>  #include <vector>    void f(std::vector<**int**> &v, **size\_t** s) {    v.resize(s); // May throw, but that is okay  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Heed Compiler Warnings, Sanitize Data Sent to Other Systems, Architect and Design for Security Policies, Use Effective Quality Assurance Techniques, and Adopt a Secure Coding Standard.  If a function declared with a noexcept-specification throws an exception of a type that would cause the noexcept-specification to evaluate to false, the function std::terminate() will be called. Calling std::terminate() leads to implementation-defined termination of the program. To prevent [abnormal termination](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-abnormaltermination) of the program, any function that declares an exception-specification should restrict itself, as well as any functions it calls, to throwing only allowed exceptions. Throwing unexpected exceptions disrupts control flow and can cause premature termination and [denial of service](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-denial-of-service). |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | unhandled-throw-noexcept | Partially checked |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 56 D | Partially implemented |
| [Parasoft C/C++Test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | CERT\_CPP-ERR55-a | Where a function's declaration includes an exception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |
| [RuleChecker](https://wiki.sei.cmu.edu/confluence/display/cplusplus/RuleChecker) | 20.10 | unhandled-throw-noexcept | Partially checked |

### 

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output | STD-888-CPP | Do not alternately input and output from a file stream without an intervening positioning call |

| **Noncompliant Code** |
| --- |
| This example appends data to the end of a file and then reads it from the same file. Consequently, because there is no position call present between the output and input calls, the behavior is undefined. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";    std::string str;    file >> str;  } |

| **Compliant Code** |
| --- |
| The solution would be to. Use std::basic\_istream<T>::seekg() function between the output and input to eradicate the undefined behavior. |
| #include <fstream>  #include <string>    void f(const std::string &fileName) {    std::fstream file(fileName);    if (!file.is\_open()) {      // Handle error      return;    }      file << "Output some data";      std::string str;    file.seekg(0, std::ios::beg);    file >> str;  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Heed Compiler Warnings, Sanitize Data Sent to Other Systems, Architect and Design for Security Policies, Use Effective Quality Assurance Techniques, Practice Defense in Depth, and Adopt a Secure Coding Standard.  When a file is opened with update mode . . ., both input and output may be performed on the associated stream. However, output shall not be directly followed by input without an intervening call to the fflush function or to a file positioning function (fseek, fsetpos, or rewind), and input shall not be directly followed by output without an intervening call to a file positioning function, unless the input operation encounters end-of-file. Alternately inputting and outputting from a stream without an intervening flush or positioning call is [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/c/BB.+Definitions#BB.Definitions-undefinedbehavior). |

**Threat Level**

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

**Automation**

| Tool | Version | Checker | Description Tool |
| --- | --- | --- | --- |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/c/LDRA) | 9.7.1 | 84 D | Fully implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/c/Parasoft) | 2020.2 | CERT\_C-FIO39-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| [PC-lint Plus](https://wiki.sei.cmu.edu/confluence/display/c/PC-lint+Plus) | 1.4 | 2478, 2479 | Fully supported |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/c/Polyspace+Bug+Finder) | R2020a | [CERT C: Rule FIO39-C](https://www.mathworks.com/help/bugfinder/ref/certcrulefio39c.html) | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

**Coding Standard 9**

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Containers | STD-999-CPP | Do not subtract iterators that do not refer to the same container |

| **Noncompliant Code** |
| --- |
| This example attempts to define whether or not the pointer test is within range of [r, r + n]. Nevertheless, when test does not point within the range, the subtraction yields undefined behavior. |
| #include <cstddef>  #include <iostream>    template <typename Ty>  **bool** in\_range(const Ty \*test, const Ty \*r, **size\_t** n) {    return 0 < (test - r) && (test - r) < (std::**ptrdiff\_t**)n;  }    void f() {  **double** foo[10];  **double** \*x = &foo[0];  **double** bar;    std::cout << std::boolalpha << in\_range(&bar, x, 10);  } |

| **Compliant Code** |
| --- |
| [This solution exhibits a fully portable implementation of in\_range() function to compare test against possible addresses in the range [r, n]. Therefore, now this solution provides both efficient and a portable. |
| #include <iostream>    template <typename Ty>  **bool** in\_range(const Ty \*test, const Ty \*r, **size\_t** n) {    auto \*cur = reinterpret\_cast<const unsigned **char** \*>(r);    auto \*end = reinterpret\_cast<const unsigned **char** \*>(r + n);    auto \*testPtr = reinterpret\_cast<const unsigned **char** \*>(test);      for (; cur != end; ++cur) {      if (cur == testPtr) {        return true;      }    }    return false;  }    void f() {  **double** foo[10];  **double** \*x = &foo[0];  **double** bar;    std::cout << std::boolalpha << in\_range(&bar, x, 10);  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Architect and Design for Security Policies, Use Effective Quality Assurance Techniques, and Adopt a Secure Coding Standard.  If two unrelated iterators (including pointers) are subtracted, the operation results in [undefined behavior](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-undefinedbehavior). Do not subtract two iterators (including pointers) unless both point into the same container or one past the end of the same container. Therefore, both iterators must refer to the same container object or must be obtained via a call to end()(or cend()) on the same container object. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Astrée](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=222953724) | 20.10 | invalid\_pointer\_subtraction invalid\_pointer\_comparison |  |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 70 S, 87 S, 437 S, 438 S | Enhanced Enforcement |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | CERT\_CPP-CTR54-a CERT\_CPP-CTR54-b CERT\_CPP-CTR54-c | Do not compare iterators from different containers Do not compare two unrelated pointers Do not subtract two pointers that do not address elements of the same array |
| [PRQA QA-C++](https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046345) | 4.4 | 2668, 2761, 2762, 2763, 2766, 2767, 2768 | Enforced by QA-CPP |

**Coding Standard 10**

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Characters and Strings | STD-1010-CPP | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| This example shows how the input is unbounded causing the code to be vulnerable and result in a buffer overflow |
| #include <iostream>    void f() {  **char** buf[12];    std::cin >> buf;  } |

| **Compliant Code** |
| --- |
| This solution ensures the data is not truncated and protected against buffer overflows using std::string instead of a bounded array. |
| #include <iostream>  #include <string>    void f() {    std::string input;    std::string stringOne, stringTwo;    std::cin >> stringOne >> stringTwo;  } |

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

|  |
| --- |
| **Principles(s):** Validate Input Data, Heed Compiler Warnings, Sanitize Data Sent to Other Systems, Architect and Design for Security Policies, Use Effective Quality Assurance Techniques, Practice Defense in Depth, and Adopt a Secure Coding Standard.  Copying data to a buffer that is not large enough to hold that data results in a buffer overflow. Buffer overflows occur frequently when manipulating strings. Therefore, attackers can [exploit](https://wiki.sei.cmu.edu/confluence/display/cplusplus/BB.+Definitions#BB.Definitions-exploit) this condition to execute arbitrary code with the permissions of the vulnerable process. To prevent such errors, either limit copies through truncation or, preferably, ensure that the destination is of sufficient size to hold the data to be copied. C-style strings require a null character to indicate the end of the string, while the C++ std::basic\_string template requires no such character. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [CodeSonar](https://wiki.sei.cmu.edu/confluence/display/cplusplus/CodeSonar) | 6.0p0 | MISC.MEM.NTERM  LANG.MEM.BO LANG.MEM.TO | No space for null terminator  Buffer overrun Type overrun |
| [LDRA tool suite](https://wiki.sei.cmu.edu/confluence/display/cplusplus/LDRA) | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| [Parasoft C/C++test](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Parasoft) | 2020.2 | CERT\_CPP-STR50-b CERT\_CPP-STR50-c CERT\_CPP-STR50-e CERT\_CPP-STR50-f CERT\_CPP-STR50-g | Avoid overflow due to reading a not zero terminated string Avoid overflow when writing to a buffer Prevent buffer overflows from tainted data Avoid buffer write overflow from tainted data Do not use the 'char' buffer to store input from 'std::cin' |
| [Polyspace Bug Finder](https://wiki.sei.cmu.edu/confluence/display/cplusplus/Polyspace+Bug+Finder) | R2020a | [CERT C++: STR50-CPP](https://www.mathworks.com/help/bugfinder/ref/certcstr50cpp.html) | Checks for:  Use of dangerous standard function  Missing null in string array  Buffer overflow from incorrect string format specifier  Destination buffer overflow in string manipulation  Rule partially covered. |

## Defense-in-Depth Illustration

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



# Project One

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

## Revise the C/C++ Standards

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

## Risk Assessment

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

## Automated Detection

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

## Automation

Provide a written explanation using the image provided.



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

"Rapid and secure code delivery" may be an oxymoron to most businesses, but DevSecOps aims to change that assumption.DevSecOps is a way of approaching IT security with an "everyone is responsible for security" mindset. It involves injecting security practices into an organization's DevOps pipeline. The goal is to incorporate security into all stages of the software development workflow. That's contradictory to its predecessor development models—DevSecOps means you're not saving security for the final stages of the SDLC. DevSecOps is based on the principle of DevOps, which will help your case for making the switch. And doing so will enable you to bring together proficient individuals from across different technical disciplines to enhance your existing security processes.

Whatever definition you prefer for DevOps, it's certain to include the concept of automation in order to ensure fast feedback throughout the DevOps pipeline. The sooner you get feedback, the quicker you can take action to resolve problems. Ideally, a DevOps process is infused with automation in order to accomplish tasks more quickly and consistently like:

1. Developing the code
2. Testing it
3. Deploying the test infrastructure
4. Running end-to-end tests
5. Deploying the code
6. Collecting metrics from production
7. End-to-end orchestration of the DevOps pipeline

Liran Levy, R&D Lifecycle DevOps Manager at HPE Software, says that the ability to reduce your time to market is directly correlated to the ability to obtain fast feedback at every stage of the pipeline, and this can only be done with automation. The range of opportunities for automation in DevOps, along with the choice of tools that you can use, can seem complex, especially for smaller teams with limited time and resources. You might already be doing some automations today, but how do you prioritize what to focus on next? If you're beginning a new DevOps project from scratch, where do you start?

**Where are you now with automation?**

Most DevOps projects start out as agile projects. If you're not there yet, make sure you get trained, and enlist an agile coach. Start working on defining your backlog, executing sprints, and producing content regularly. The goal with agile development and particularly with DevOps is continuous delivery.

JezHumble, co-author of the book "Continuous Delivery" and UC Berkeley lecturer, has two golden rules for continuous delivery:

1. Software (trunk) is always deployable.
2. Everyone is checking into trunk daily (at least)

He cites three ingredients in achieving these golden rules: configuration management, continuous integration, and automated testing. Once you're working according to agile principles, your developers should be using configuration management, doing continuous integration, and running automated unit tests as part of the build. Automated functional tests, with perhaps some manual ones as well, are being developed and run concurrently with development. There might also be some non-functional tests, such as performance and security tests. The automated tests might be configured to run automatically, but some may need to be kicked off manually. The build process is probably configured to package the software so that it can be deployed for further testing.

So, if you're doing all that, what automation do you need to add to get to DevOps?

1. Defend the source base from check-ins that break the build, so that the code is always releasable
2. Conversion of existing manual tests to automated tests
3. Provisioning of a test environment
4. Execution of functional and non-functional tests
5. Deployment of the code
6. Rolling back a deployment to the last working version
7. Gathering production metrics
8. Orchestrating the pipeline
9. Continuously updating the DevOps dashboard

As Humble and David Farley say in their book, "Continuous Delivery", you don't need to automate everything at once, and you can, and should, automate gradually, over time.

4 steps to help you prioritize DevOps automation

Here are four simple steps to prioritizing your automation.

**1. Evaluate your needs:**

The first step is to understand what you want to achieve, so that you can focus on your goals. You might already have some of these in place, although they might need a little more investment:

Release software changes frequently into production

Get quick feedback at any stage of the pipeline about a change before it goes into production

Reduce the cost of downtime

Monitor the software in production

Have a single status report for the whole team by feeding production metrics back to a single dashboard for developers and operations staff

This isn't an exhaustive list. But once you have your own list, you need to prioritize it, and then work out what's blocking you from achieving each of the goals.

**2. Identify your bottlenecks:**

Once you've identified the most important goal to achieve, you need to diagnose the reason why you've not achieved it yet so that you can treat it. For example, if your primary goal is to release small software changes frequently into production, there are many possible reasons that you're not there yet:

1. Do your developers often cause delay by checking in bad code and breaking the build?
2. Do your developers need to take time out to explain to the testers how to deploy software changes for testing?
3. Does it take your testers a long time to test a small change?
4. Do you need to manually prepare and load data for testing?

Once you've found the bottlenecks, think about the reasons they exist:

1. If you have build issues, you need to be using continuous integration. Use a gated check-in solution, to prevent code getting to the main branch if it's going to break the build.
2. Are you spending a lot of time testing? If you're running the same manual tests over and over again, automate them. If you're spending most of your time setting up test environments, automate the provisioning of test machines and deploying the software to be tested. If preparing the data is the problem, automate the process of extracting and scrubbing data, and populating the database.
3. If it takes time to begin testing because a tester needs to start the tests manually, the tests need to be configured to run automatically as soon as the build succeeds, and the test environment is available.
4. Testers and developers work very closely together as part of the same team, but developers must do sufficient testing of their own work before allowing others to test it.
5. If you need to shut your production environment down every time you find a problem in a new change, you should automate the process of reverting the build back to a stable version, and make sure that this process is tested before the change is deployed.

**3. Consider the return on investment:**

By this point, you have created a list of tasks that you can add to your project's backlog. This lets you prioritize them relative to your other backlog items, and affords full visibility of the automation improvements. You know your team's capacity, and by including these automation items in the backlog, you ensure that these 'support' activities are being factored into the team's workload.

As with any backlog item, large items can be broken down into smaller items, and implemented in stages, with each stage providing value. Evaluate the cost of implementing each item, and by estimating the relative cost of delaying their implementation, you can use a weighted shortest job first approach to compare the return on investment of each item.

The advantage of this approach is that you can gradually phase your automation improvements into the backlog while minimizing delay in implementing the rest of your backlog items.

Keep in mind though that test automation should be part of the definition of done for all new code. John Jeremiah, a digital researcher for Hewlett Packard Enterprise, puts it very clearly: "In the old world, automation is 'nice to have', or 'support'. In the new world, automation is essential to achieving velocity without sacrificing quality. In fact, NO new code should be written without the automation to test it."

**4. Continuously assess and reevaluate:**

Once you've cleared one of your bottlenecks, you can move on to the next one. But circumstances may have changed since you last investigated what was holding up your DevOps pipeline, so you need to reexamine the pipeline. There might be a new bottleneck that needs to be resolved, or perhaps an existing one has become more critical. Levy says that constantly reevaluating the pipeline and prioritizing the most serious hold-up alongside the rest of the items to be developed, will lead to an efficient DevOps pipeline, but you need to keep re-evaluating throughout the life of the project.

**DevOps automation = Continuous monitoring and improving**

DevOps automation isn't something that is done once and forgotten about. Just like security and performance, it's something you need to continuously monitor and improve as long as the DevOps pipeline is still active.

The first step to DevOps automation is to be agile. Once you're there, you can start on a gradual journey towards full-blown DevOps. Look at the DevOps goals that you want to achieve and work out what's stopping you from getting there. Once you know what the bottlenecks are, you can figure out what you need to do to resolve them and make that part of your backlog. If you continuously prioritize and reevaluate these items relative to the other items on the backlog, you'll be able to introduce automation gradually and effectively.

## Summary of Risk Assessments

Consolidate all risk assessments into one table including both coding and systems standards, ordered by standard number.

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | High | Unlikely | Medium | High | 2 |
| STD-111-C | Low | Unlikely | High | P1 | L3 |
| STD-222-C | High | Probable | High | P6 | L2 |
| STD-333-C | Medium | Unlikely | Medium | P4 | L3 |
| STD-444-C | High | Likely | Medium | P18 | L1 |
| STD-555-C | High | Likely | Medium | P18 | L1 |
| STD-666-C | Low | Unlikely | High | P1 | L3 |
| STD-777-CPP | Low | Likely | Low | P9 | L2 |
| STD-888-CPP | Low | Likely | Medium | P6 | L2 |
| STD-999-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-1010-CPP | High | Likely | Medium | P18 | L2 |

## Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Designed to prevent the attacker from accessing the unencrypted data by ensuring the data is encrypted when on disk. The policy applies to encrypt data at the requisite times. |
| Encryption at flight | The process of encrypting data (scrambling readable text so it can be only read by the person who has the secret code or decryption key) while the data is being transmitted. The policy applies to provide data security for sensitive information. |
| Encryption in use | It’s a capability that lets you run your computation on encrypted data or run encrypted applications. The policy applies to protect information in transit between authentication from the requested system. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Provides a way of identifying a user, typically by having the user enter a valid username and valid password before access is granted. The policy applies to check a user’s credentials to see if it’s a match and if not, network access is denied to protect the system. |
| Authorization | The authorization process determines whether the user has the authority to issue such commands. Authorization policy applies to enforce certain policies such as determining what types or qualities of activities, resources, or services a user is permitted. |
| Accounting | Accounting measures the resources a user consumes during access. This can include the amount of system time or the amount of data a user has sent and/or received during a session. Accounting policy applies by logging session statistics and usage information and is used for authorization control, billing, trend analysis, resource utilization, and capacity planning activities. |

**\***Use this checklist for the Triple A to be sure you include these elements in your policy:

* User logins
* Changes to the database
* Addition of new users
* User level of access
* Files accessed by users

## Map the Principles

Map the principles to each of the standards, and provide a justification for the connection between the two. In the Module Three milestone, you added definitions for each of the 10 principles provided. Now it’s time to connect the standards to principles to show how they are supported by principles. You may have more than one principle for each standard, and the principles may be used more than once. Principles are numbered 1 through 10. You will list the number or numbers that apply to each standard, then explain how each of these principles supports the standard. This exercise demonstrates that you have based your security policy on widely accepted principles. Linking principles to standards is a best practice.

**NOTE:** Green Pace has already successfully implemented the following:

* Operating system logs
* Firewall logs
* Anti-malware logs

The only item you must complete beyond this point is the Policy Version History table.

# Audit Controls and Management

Every software development effort must be able to provide evidence of compliance for each software deployed into any Green Pace managed environment.

Evidence will include the following:

* Code compliance to standards
* Well-documented access-control strategies, with sampled evidence of compliance
* Well-documented data-control standards defining the expected security posture of data at rest, in flight, and in use
* Historical evidence of sustained practice (emails, logs, audits, meeting notes)

# Enforcement

The office of the chief information security officer (OCISO) will enforce awareness and compliance of this policy, producing reports for the risk management committee (RMC) to review monthly. Every system deployed in any environment operated by Green Pace is expected to be in compliance with this policy at all times.

Staff members, consultants, or employees found in violation of this policy will be subject to disciplinary action, up to and including termination.

# Exceptions Process

Any exception to the standards in this policy must be requested in writing with the following information:

* Business or technical rationale
* Risk impact analysis
* Risk mitigation analysis
* Plan to come into compliance
* Date for when the plan to come into compliance will be completed

Approval for any exception must be granted by chief information officer (CIO) and the chief information security officer (CISO) or their appointed delegates of officer level.

Exceptions will remain on file with the office of the CISO, which will administer and govern compliance.

# Distribution

This policy is to be distributed to all Green Pace IT staff annually. All IT staff will need to certify acceptance and awareness of this policy annually.

# Policy Change Control

This policy will be automatically reviewed annually, no later than 365 days from the last revision date. Further, it will be reviewed in response to regulatory or compliance changes, and on demand as determined by the OCISO.

# Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 3/18/2021 | Milestone Three: Coding Standards | Janera Dobson | SNHU |
| 3.0 | 4/09/2021 | Project One: Security Policy | Janera Dobson | SNHU |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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

Reference:

Schiela, R. (2020, May 29). SEI CERT C++ Coding Standard. Retrieved March 17, 2021, from <https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682>.