**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 | Covers that the input should be validated when coming from an untrusted source. We should be cautious when considering outside data sources. This includes environment variables, network interfaces, command line arguments, and files uploaded by users. |
| 1. Heed Compiler Warnings | We should be using the highest warning level that applies to our compiler to assist in getting rid of warnings when writing the code. Additional static and dynamic analysis tools can also be used to find and remove additional security flaws. |
| 1. Architect and Design for Security Policies | Aim to write and design software to use and follow the required security policies that are in place. One way to do this is to divide the system into sections and set privileges depending on which section that particular system is communicating with. |
| 1. Keep It Simple | Complex code is more likely to contain errors when creating and once the code is put in use. It is best practice to write code to be short and as simple as possible. |
| 1. Default Deny | Access should be based on permission. Access is not granted unless certain conditions are met, such as a set username and password. |
| 1. Adhere to the Principle of Least Privilege | All processes should run with the least set of privileges that are needed to complete the process. Anything that needs more privileges will be allowed, but for the least amount of time it takes to complete the process. |
| 1. Sanitize Data Sent to Other Systems | When passing data between systems, all must be sanitized. Such as making sure a NULL is not a valid input. |
| 1. Practice Defense in Depth | Use defense strategies in layer, so that if one layer fails, another is in place to protect from vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Use quality assurance techniques to find and remove vulnerabilities. Techniques could consist of code audits and reviews done by external parties. |
| 1. Adopt a Secure Coding Standard | Depending on the development language and platform, it is important to create or apply a secure coding standard when developing software. |

(Schiela & 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] | Must declare identifiers, variables, and functions before utilizing. |

| **Noncompliant Code** |
| --- |
| Unspecified Implicit int |
| Extern foo; |

| **Compliant Code** |
| --- |
| Specified Implicit int |
| Extern int foo; |

(Razmyslov & Seacord, 2021)

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

| **Principles(s):**   1. Validate input data: only accept input that is expected |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| TrustInSoft Analyzer | 1.38 | Type specifier missing | Checks if things are partially verified and detects undefined behavior. |
| SonarQube C/C++ Plugin | 3.11 | S819, S820 | Checks for partially implemented; implicit return type not covered. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Do not read uninitialized int/variables |

| **Noncompliant Code** |
| --- |
| [Noncompliant description]  Uninitialized local variable is evaluated |
| #include <iostream>  Void f() {  Int i;  Std::cout << i;  } |

| **Compliant Code** |
| --- |
| [Compliant description]  Initialize object before printing the value |
| #include <iostream>  Void f() {  Int i = 0;  Std::cout << i;  } |

(Pincar & Razmyslov, 2021)

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

| **Principles(s):**   1. Validate Input Data: initialize all variables to help prohibit the execution of code within the program and unexpected behavior. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | LANG.STRUCT.RPL  LANG.MEM.UVAR | Return pointer to local uninitialized variable |
| Polyspace Bug Finder | R2021b | CERT C++: EXP53-CPP | Checks for non-initialized variable and non-initialized pointer. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Make sure that storage for string object has ample space char data and null terminator. |

(Britton & Pincar, 2021)

| **Noncompliant Code** |
| --- |
| Input is not bound, causing a buffer overflow |
| #include <iomanip>  #include <iostream>  int main()  {  std::cout << "Buffer Overflow Example" << std::endl;  const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::cout << "Enter a value: ";  std::cin >> user\_input;  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl;  } |

| **Compliant Code** |
| --- |
| When using the std::setw() function, the output stream was manipulated to set a width parameter, stopping the overflow |
| #include <iomanip>  #include <iostream>  int main()  {  std::cout << "Buffer Overflow Example" << std::endl;  const std::string account\_number = "CharlieBrown42";  char user\_input[20];  std::cout << "Enter a value: ";    // use a setw function to manipulate the output stream to set a width parameter  std::cin >> std::setw(20) >> user\_input;  std::cout << "You entered: " << user\_input << std::endl;  std::cout << "Account Number = " << account\_number << std::endl;  } |

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

| **Principles(s):**   1. Validate input data: making sure enough space for data helps to prevent overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 489 S, 66 X, 70 X, 71 X | Partially implemented |
| CodeSonar | 6.2p0 | MISC.MEM.NTERM  LANG.MEM.BO  LANG.MEM.TO | No space for null terminator  Buffer overrun  Type overrun |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | Sanitize data that is passed from different systems. |

| **Noncompliant Code** |
| --- |
| The call passes unsanitized data from an untrusted source (USER environment variable). |
| (void) execl(LOGIN\_PROGRAM, “login”’  “-p”,  “-d”, slavename,  “-h”, host,  “-s”, pam\_svc\_name,  (AuthenticatingUser != NULL ? AuthenticatingUser :  Getenv(“USER”)),  0); |

| **Compliant Code** |
| --- |
| The program uses POSIX getopt() function to parse cmd line arguments. Inserting the “—” argument before the call causes getopt() to stop interpreting options in the argument list. So, the USER variable cannot be used by an attacker to inject an additional command-line option. |
| (void) execl(LOGIN\_PROGRAM, “login”’  “-p”,  “-d”, slavename,  “-h”, host,  “-s”, pam\_svc\_name,  “--”,  (AuthenticatingUser != NULL ? AuthenticatingUser :  Getenv(“USER”)),  0); |

(Herter & Seacord, 2020)

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

| **Principles(s):**  7) Sanitize Data Sent to Other Systems: sanitizing data helps to prevent the use of SQL, command, and other injection attack methods. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | 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 |
| Polyspace Bug Finder | R2021b | CERT C++: EXP53-CPP | Checks for non-initialized variable and non-initialized pointer. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Freed memory should not be accessed. |

| **Noncompliant Code** |
| --- |
| P is free before s-> next is executed, so p->next reads memory that’s 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** |
| --- |
| To fix, a reference is stored 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);  }  } |

(Seacord & Razmyslov, 2021)

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

| **Principles(s):**   1. Validate input data: Reading un-initialized variables can cause the program to behave improperly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2021b | CERT C++: EXP53-CPP | Checks for non-initialized variable and non-initialized pointer. |
| Clang | 3.9 | -Wuninitialized  clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Use assertions to test values of constant expressions. |

| **Noncompliant Code** |
| --- |
| Assert() is used to assert a property concerning a memory mapped structure that is needed for the program to run as intended. |
| #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** |
| --- |
| Assertions regarding constant expressions, a preprocessor conditional statement is ok to use. |
| struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };    #if (sizeof(struct timer) != (sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int)))  #error "Structure must not have any padding"  #endif |

(Britton & Seacord, 2018)

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

| **Principles(s):**  9) Use Effective Quality Assurance Techniques: Use assertions to test values as an extra step to be sure program is behaving as intended. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | Misc-static-assert | Checked by clang-tidy |
| CodeSonar | 6.2p0 | (customization) | Users can implement a custom check that reports uses of the assert() macro |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Handle exceptions thrown prior to the main() functions is executed. |

| **Noncompliant Code** |
| --- |
| struct S might throw an exception that is not caught when globalS is built when the program is starting up. |
| struct B {  B() noexcept(false);  };    static B globalB; |

| **Compliant Code** |
| --- |
| globalB is a local variable with static storage duration. This lets any exceptions thrown during object construction to be caught. This happens because the constructor for B will be executed the first time the function globalB() is called instead of being ran at program startup |
| struct B {  B() noexcept(false);  };    B &globalB() {  try {  static B b;  return b;  } catch (...) {  // Handle error, perhaps by logging it and gracefully terminating the application.  }  // Unreachable.  } |

(Gangopadhyay & Seacord, 2021)

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

| **Principles(s):**  3) Architect and Design for Security Policies: exceptions that can’t be caught can make the program terminate unexpectantly can lead to denial of service attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Polyspace Bug Finder | R2021b | CERT C++: ERR58-CPP | Checks for exceptions raised during program start up |
| RuleChecker | 20.10 | Potentially-throwing-static-initialization | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Containers** | [STD-008-CPP] | Use valid iterators to reference the proper elements of the container. |

| **Noncompliant Code** |
| --- |
| Pos is invalidated after the first call to the insert() function and the following loop iterations have undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> p;  auto pos = p.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  p.insert(pos, items[i] + 41.0);  }  } |

| **Compliant Code** |
| --- |
| Pos is assigned a valid iterator on every insertion, which will avert undefined behavior. |
| #include <deque>    void f(const double \*items, std::size\_t count) {  std::deque<double> p;  auto pos = p.begin();  for (std::size\_t i = 0; i < count; ++i, ++pos) {  pos = p.insert(pos, items[i] + 41.0);  }  } |

(Long & O’Donnell, 2021)

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

| **Principles(s):**  3) Architect and Design for Security Policies: use proper iterators to prevent overflow. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.2p0 | ALLOC.UAF | Use after free |
| Parasoft C/C++test | 2021.2 | CERT\_CPP-CTR51-a | Do not modify container while iterating over it. |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Management** | [STD-009-CPP] | Make sure there is adequate memory for an object. |

| **Noncompliant Code** |
| --- |
| Not enough space is allocated for a struct time object. The size of the pointer is being used to determine the size of the pointed-to object. |
| #include <stdlib.h>  #include <time.h>    struct time \*make\_time(int year, int mon, int day, int hour,  int min, int sec) {  struct time \*timeb;  timeb = (struct time \*)malloc(sizeof(timeb));  if (timeb == NULL) {  return NULL;  }  \*timeb = (struct tm) {  .time\_sec = sec, .time\_min = min, .time\_hour = hour,  .time\_mday = day, .time\_mon = mon, .time\_year = year  };  return timeb;  } |

| **Compliant Code** |
| --- |
| Now the right amount of memory is saved for the constructor time object. This is done by passing the pointer type to the size of operator. |
| #include <stdlib.h>  #include <time.h>    struct time \*make\_time(int year, int mon, int day, int hour,  int min, int sec) {  struct time \*timeb;  tmb = (struct time \*)malloc(sizeof(\*timeb));  if (timeb == NULL) {  return NULL;  }  \*timeb = (struct time) {  .time\_sec = sec, .time\_min = min, .time\_hour = hour,  .time\_mday = day, .time\_mon = mon, .time\_year = year  };  return timeb;  } |

(Britton & Gennari, 2021)

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

| **Principles(s):**  3) Architect and Design for Security Policies: invalid size arguments to memory functions can cause overflows, which can lead to vulnerabilities. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Detects undefined behavior |
| Parasoft C/C++test | 2021.2 | CERT\_C-MEM35-a | Do not use sizeof operator on pointer type to specify the size of the memory to be allocated using “malloc”, “calloc”, or “realloc” functions. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input Output** | [STD-010-CPP] | Closed files should not be accessed. |

| **Noncompliant Code** |
| --- |
| If you use a FILE after the file has been closed, will cause undefined behavior. For the following example, the stdout stream is used after it has been closed. |
| #include <stdio.h>    int close\_stdout(void) {  if (fclose(stdout) == EOF) {  return -1;  }    printf("stdout successfully closed.\n");  return 0;  } |

| **Compliant Code** |
| --- |
| Stdout isn’t used after it is closed. Using fputs() writes the string to the output stream. |
| #include <stdio.h>    int close\_stdout(void) {  if (fclose(stdout) == EOF) {  return -1;  }    fputs("stdout successfully closed.", stderr);  return 0;  } |

(Britton & Seacord, 2018)

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

| **Principles(s):**  8) Practice Defense in Depth: making sure closed files are not accessible adds another layer of defense. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++test | 2021.2 | CERT\_C-FIO46-a | Do not use resources that have been freed. |
| Polyspace Bug Finder | R2021a | CERT C: Rule FIO46-C | Checks for use of previously closed resource. |

### 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.

When looking at the left side if the DevSecOps diagram we first look at the Assess and plan section. Here is where we aim to write and design software to use and follow the required security policies that are in place. Here is where we look at what potential vulnerabilities could be. If we move to Build section, we will refer to our STD-006. By implementing assertions, we can test that the system performs as intended. This add an additional layer of protection. Looking at the right side of the diagram we can add additional tools to monitor and detect section. These tools would run scans to check for vulnerabilities. We will also keep automated logs to keep track of what’s going on within the system. By adding more layers of defense in this section, it insures that we are able to stop a problem before it happens.

### Summary of Risk Assessments

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

| **Rule** | **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- | --- |
| STD-001-CPP | Low | Unlikely | Medium | P3 | L3 |
| STD-002-CPP | High | Probable | Medium | P12 | L1 |
| STD-003-CPP | High | Likely | Medium | P18 | L1 |
| STD-004-CPP | High | Likely | Medium | P18 | L1 |
| STD-005-CPP | High | Probable | Medium | P12 | L1 |
| STD-006-CPP | Low | Unlikely | High | P1 | L3 |
| STD-007-CPP | Low | Likely | Low | P9 | L2 |
| STD-008-CPP | High | Probable | High | P6 | L2 |
| STD-009-CPP | High | Probable | High | P6 | L2 |
| STD-010-CPP | Medium | Unlikely | Medium | P4 | L3 |

### Create Policies for Encryption and Triple A

Include all three types of encryptions (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 | Encryption in rest is is done when the data is not being actively used. It encrypts the data where it is stored, including hard drives, laptops, flash drives, or cloud storage. This protects the data from being accessed by cybercriminals. There are multiple companies that offer data encryption software such as LastPass, Bitlocker, 7-Zip, and AxCrypt (ENCRYPTION: UNDERSTANDING DATA AT REST VS. IN TRANSIT, 2021). |
| Encryption at flight/transit | Encryption at flight/transit is encryption done while the data is active and moving between devices and networks. For file transfers use SPC (Secure Copy Protocol), SFTP (Secure File Transfer Protocol), or over Remote Desktop when transferring to other networked devices. For web servers, use TLS(transport layer security) (OpsCompass Staff, 2015). |
| Encryption in use | Encryption in use is encryption being done when the data is being processed, accessed, or read. To secure the data in use access will be granted by user role. This will limit the access to certain individuals. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Who wants access? Users will have to use their credentials (user name and password) to access different systems. Users will be challenged by asking for those credentials. This ensures that access is not granted to users without the proper credentials |
| Authorization | What is the user allowed to access? Authorizations/permissions will be set for each networked user. This will control what access a user will have depending on the role of the user. This ensures that only users with the proper authorization will be able to access or modify objects within our network. |
| Accounting | Who did what and when? Accounting refers to monitoring what is going on within our system. This includes what events take place, the resources being used, and keeping logs to track information on the events. |

**\***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 | 4/10/2022 | Amendment/Final Draft | Xavier Sepulveda | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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

References

Britton, J. & Gennari, J. (2021, November 30). *MEM35-C. Allocate sufficient memory for an object.* Retrieved from <https://wiki.sei.cmu.edu/confluence/display/c/MEM35-C.+Allocate+sufficient+memory+for+an+object>

Britton, J. & Pincar, J. (2021, December 09). *STR-CPP. Guarantee that storage for strings has sufficient space for character data and the null terminator*. Retrieved from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR50-CPP.+Guarantee+that+storage+for+strings+has+sufficient+space+for+character+data+and+the+null+terminator>

Britton, J & Seacord, R. (2018, December 11). *DLC03-C. Use a static assertion to test the value of a constant expression*. Retrieved from <https://wiki.sei.cmu.edu/confluence/display/c/DCL03-C.+Use+a+static+assertion+to+test+the+value+of+a+constant+expression>

Britton, J & Seacord, R. (2021, April 23). *FIO46-C. Do not access a closed file.* Retrieved from <https://wiki.sei.cmu.edu/confluence/display/c/FIO46-C.+Do+not+access+a+closed+file>

ENCRYPTION: UNDERSTANDING DATA AT REST VS. IN TRANSIT. (2021, September 01). Retrieved from <https://datalocker.com/blog/technology/encryption/encryption-at-rest-vs-in-transit-effectively-encrypt-identifiable-information/>

Gangopadhyay, A & Seacord, J. (2021, July 19). *ERR58-CPP. Handle all exceptions thrown before main() begins executing*. Retrieved from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/ERR58-CPP.+Handle+all+exceptions+thrown+before+main%28%29+begins+executing>

Herter, J & Seacord, R. (2020, August 13). *STR02-C. Sanitize data passed to complex subsystems*. Retrieved from <https://wiki.sei.cmu.edu/confluence/display/c/STR02-C.+Sanitize+data+passed+to+complex+subsystems>

Long, F. & O’Donnell, J. (2021, December 29). *CTR51-CPP. Use valid references, pointers, and iterators to reference elements of a container.* Retrieved from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/CTR51-CPP.+Use+valid+references%2C+pointers%2C+and+iterators+to+reference+elements+of+a+container>

OpsCompass Staff. (2015, July 10). Are you encrypting you data-in-flight? If not, you should be. Retrieved from <https://opscompass.com/resources/blog/are-you-encrypting-your-data-in-flight-if-not-you-should-be/#:~:text=Encrypting%20data%20in%2Dflight%20means,being%20transmitted%20over%20a%20network>.

Pincar, J. & Razmyslov, S. (2021, June 25). *EXP53-CPP. Do not read uninitialized memory*. Retrieved from <https://wiki.sei.cmu.edu/confluence/display/cplusplus/EXP53-CPP.+Do+not+read+uninitialized+memory>

Razmyslov, S. & Seacord, R. (2021, June 24). *DCL31-C. Declare identifiers before using them*. Retrieved from <https://wiki.sei.cmu.edu/confluence/display/c/DCL31-C.+Declare+identifiers+before+using+them>

Razmyslov, S. & Seacord, R. (2021, June 24). MEM30-C. Do not access freed memory. Retrieved from <https://wiki.sei.cmu.edu/confluence/display/c/MEM30-C.+Do+not+access+freed+memory>

Schiela, R. & Seacord, R. (2018, May 02). *Top 10 Secure Coding Practices*. Retrieved from <https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices>

“Protect your network” with patch management, auditing, and security scanning. (2022, April 12). *Protect your network with patch management, auditing, and security scanning.* Retrieved from <https://www.gfi.com/products-and-solutions/network-security-solutions/gfi-languard>