**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 | Input validation is where you validate input from any untrusted sources of data. Validating input can reduce and/or eliminate a vast number of vulnerabilities in the software. |
| 1. Heed Compiler Warnings | Run code with the highest level of warning available for the compiler and modify the code accordingly to try to eliminate the warnings. |
| 1. Architect and Design for Security Policies | Enforce security policies and create an architecture and design to follow. |
| 1. Keep It Simple | Keep the design as small and simple as possible. This will reduce the errors made in the configuration, use, and configuration. Also, as more security techniques become complex, assurance increases. |
| 1. Default Deny | Access needs to be based on permissions and not exclusions. Have access be denied at the start and validate conditions in order to grant permissions. |
| 1. Adhere to the Principle of Least Privilege | Every operation should run with the least number of privileges to do the job correctly. If secured data is required, it should only be accessed for the least amount of time necessary. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all the data that is to be passed to the subsystem. Attackers might be able to invoke functionality that is unused within the system via injection or other methods of attacks. |
| 1. Practice Defense in Depth | Protect the system with multiple layers of security that overlap. This way if one layer is bypassed, there are still other layers of protection to prevent security vulnerabilities. |
| 1. Use Effective Quality Assurance Techniques | Quality assurance methods can be efficient in eliminating/recognizing vulnerabilities within the system. |
| 1. Adopt a Secure Coding Standard | Originate and apply secure coding techniques for the target language the system will be implemented in and platform of that system. |

### 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-INT35-C | Use correct integer precisions |

| **Noncompliant Code** |
| --- |
| This example is a function that produces 2 raised to the power of the argument of the function. The function ensures that the argument is less than the number of bits used to store a value of unsigned int. |
| #include <limits.h>    unsigned int pow2(unsigned int exp) {  if (exp >= sizeof(unsigned int) \* CHAR\_BIT) {  /\* Handle error \*/  }  return 1 << exp;  } |

| **Compliant Code** |
| --- |
| The solution uses popcount() function, which will count the number of bits set on any unsigned int which will allow the code to determine the precision of any int whether it be signed or unsigned. |
| #include <stddef.h>  #include <stdint.h>    /\* Returns the number of set bits \*/  size\_t popcount(uintmax\_t num) {  size\_t precision = 0;  while (num != 0) {  if (num % 2 == 1) {  precision++;  }  num >>= 1;  }  return precision;  }  #define PRECISION(umax\_value) popcount(umax\_value) |

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

| **Principles(s):** Use Effective Quality Assurance Techniques – this vulnerability maps to this standard because if the  the size isn’t big enough to fit the integer, then the data might be incorrect |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | N/A | Supported: Astrée reports overflows due to insufficient precision. |
| CodeSonar | 8.1p0 | LANG.ARITH.BIGSHIFT | Shift Amount Exceeds Bit Width |
| Helix QAC | 2024.1 | C0582  C++3115 | N/A |
| Parasoft C/C++test | 2023.1 | CERT\_C-INT35-a | Use correct integer precisions when checking the right hand operand of the shift operator |
| Polyspace Bug Finder | R2024a | CERT C: Rule INT35-C | Checks for situations when integer precisions are exceeded (rule fully covered) |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-FLP30-C | Do not use floating-point variables as loop counters |

| **Noncompliant Code** |
| --- |
| In this example, a floating-point variable is used as a loop counter. The decimal is a repeat fraction in binary and cant be correctly represented as a binary floating-point number. |
| void func(void) {  for (float x = 0.1f; x <= 1.0f; x += 0.1f) {  /\* Loop may iterate 9 or 10 times \*/  }  } |
| In this example, a floating-point loop counter is incremented by an amount that is too small to change value given its precision. |
| void func(void) {  for (float x = 100000001.0f; x <= 100000010.0f; x += 1.0f) {  /\* Loop may not terminate \*/  }  } |

| **Compliant Code** |
| --- |
| In this compliant code example, the loop counter is an integer from which the floating point is derived. |
| #include <stddef.h>    void func(void) {  for (size\_t count = 1; count <= 10; ++count) {  float x = count / 10.0f;  /\* Loop iterates exactly 10 times \*/  }  } |
| In this example, the loop counter is an int from which the floating point value is derived. |
| void func(void) {  for (size\_t count = 1; count <= 10; ++count) {  float x = 100000000.0f + (count \* 1.0f);  /\* Loop iterates exactly 10 times \*/  }  } |

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

| **Principles(s):** Keep it simple – this vulnerability maps to this principle because the incorrect data type is used as a loop counter. Keeping it simple will allow the application to become smoother when running it. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Low | P6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | For-loop-float | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FLP30 | Fully implemented |
| Clang | 3.9 | Cer-flp30-c | Checked by clang-tidy |
| CodeSonar | 8.1p0 | LANG.STRUCT.LOOP.FPC | Float-type loop counter |
| Compass/ROSE |  |  |  |
| Coverity | 2017.07 | MISRA C 2004 Rule 13.4  MISRA C 2012 Rule 14.1 | Implemented |
| ECLAIR | 1.2 | CC2.FLP30 | Fully implemented |
| Helix QAC | 2024.1 | C3339, C3340, C3342  C++4234 |  |
| Klocwork | 2024.1 | MISRA.FOR.COUNTER.FLT |  |
| LDRA tool suite | 9.7.1 | 39S | Fully implemented |
| Parasoft  C/C++test | 2023.1 | CERT\_C-FLP30-a | Do not use floating point variables as loop counters |
| PC-Init Plus | 1.4 | 9009 | Fully supported |
| Polysource Bug Finder | R2024a | CERT: C Rule FLP30-C | Checks for use of float variable as loop counter (rule fully covered) |
| PVS-Studio | 7.31 | V1034 |  |
| RuleChecker | 24.04 | For-loop-float | Fully checked |
| SonarQube C/C++ Plugin | 3.11 | S2193 | Fully implemented |
| TrustInSoft Analyzer | 1.38 | Non-terminating | Exhaustively detects non-terminating statements (see one compliant and one non-compliant example). |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-STR31-C | Guarantee that storage for strings has sufficient space for character data and the null terminator |

| **Noncompliant Code** |
| --- |
| This example demonstrates an off-by-one error. The loop copies data from src to dest; However, the loop doesn’t account for the null terminator character. Therefore it may be incorrectly written 1 byte past the end of the dest. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |
| The gets() reads characters from the stdin to a destination array until the end-of-file terminator is reached or new line character is read. The new line character is discarded and null characters are written after the last character read into the destination array |
| #include <stdio.h>    #define BUFFER\_SIZE 1024    void func(void) {  char buf[BUFFER\_SIZE];  if (gets(buf) == NULL) {  /\* Handle error \*/  }  } |
| This uses getchar() function to read one character at a time from stdin instead of reading the entire line at once. New line characters are discarded and null characters are written after the last character read into the array. |
| #include <stdio.h>    enum { BUFFERSIZE = 32 };    void func(void) {  char buf[BUFFERSIZE];  char \*p;  int ch;  p = buf;  while ((ch = getchar()) != '\n' && ch != EOF) {  \*p++ = (char)ch;  }  \*p++ = 0;  if (ch == EOF) {  /\* Handle EOF or error \*/  }  } |
| In this example, the call to fscanf() can result in a write outside the character array buf. |
| #include <stdio.h>    enum { BUF\_LENGTH = 1024 };    void get\_data(void) {  char buf[BUF\_LENGTH];  if (1 != fscanf(stdin, "%s", buf)) {  /\* Handle error \*/  }    /\* Rest of function \*/  } |
| Command line arguments are passed to main() as pointers to strings in the array members argv[0] through argv [argc – 1]. |
| #include <string.h>    int main(int argc, char \*argv[]) {  /\* Ensure argv[0] is not null \*/  const char \*const name = (argc && argv[0]) ? argv[0] : "";  char prog\_name[128];  strcpy(prog\_name, name);    return 0;  } |
| The getenv function searches an environment list, provided by the host environment, for a string that matches the string which it points to by name. |
| #include <stdlib.h>  #include <string.h>    void func(void) {  char buff[256];  char \*editor = getenv("EDITOR");  if (editor == NULL) {  /\* EDITOR environment variable not set \*/  } else {  strcpy(buff, editor);  }  } |
| In this example, name refers to an external string. The program constructs a name of a file from the string preparation for opening the file. |
| #include <stdio.h>    void func(const char \*name) {  char filename[128];  sprintf(filename, "%s.txt", name);  } |

| **Compliant Code** |
| --- |
| In this example, the loop termination condition is modified to account for the null-terminator character that is appended to dest. |
| #include <stddef.h>    void copy(size\_t n, char src[n], char dest[n]) {  size\_t i;    for (i = 0; src[i] && (i < n - 1); ++i) {  dest[i] = src[i];  }  dest[i] = '\0';  } |
| The fgets() function reads, at most, one less than the specified number of characters from a stream into an array. This solution is compliant due to the number of characters copied from stdin to buf cannot exceed the memory allocated. |
| #include <stdio.h>  #include <string.h>    enum { BUFFERSIZE = 32 };    void func(void) {  char buf[BUFFERSIZE];  int ch;    if (fgets(buf, sizeof(buf), stdin)) {  /\* fgets() succeeded; scan for newline character \*/  char \*p = strchr(buf, '\n');  if (p) {  \*p = '\0';  } else {  /\* Newline not found; flush stdin to end of line \*/  while ((ch = getchar()) != '\n' && ch != EOF)  ;  if (ch == EOF && !feof(stdin) && !ferror(stdin)) {  /\* Character resembles EOF; handle error \*/  }  }  } else {  /\* fgets() failed; handle error \*/  }  } |
| The gets\_s() function reads, at most, one less than the number of characters specified from the stream pointed to by stdin into an array. |
| #define \_\_STDC\_WANT\_LIB\_EXT1\_\_ 1  #include <stdio.h>    enum { BUFFERSIZE = 32 };    void func(void) {  char buf[BUFFERSIZE];    if (gets\_s(buf, sizeof(buf)) == NULL) {  /\* Handle error \*/  }  } |
| The getline() function is similar to fgets() but can dynamically allocate memory for the input buffer. If successful, the getline() function returns the number of characters read, which can be used to determine if the input has any null characters prior to the new line. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    void func(void) {  int ch;  size\_t buffer\_size = 32;  char \*buffer = malloc(buffer\_size);    if (!buffer) {  /\* Handle error \*/  return;  }    if ((ssize\_t size = getline(&buffer, &buffer\_size, stdin))  == -1) {  /\* Handle error \*/  } else {  char \*p = strchr(buffer, '\n');  if (p) {  \*p = '\0';  } else {  /\* Newline not found; flush stdin to end of line \*/  while ((ch = getchar()) != '\n' && ch != EOF)  ;  if (ch == EOF && !feof(stdin) && !ferror(stdin)) {  /\* Character resembles EOF; handle error \*/  }  }  }  free (buffer);  } |
| In this example, characters are no longer copied to buf once index == BUFFERSIZE – 1, leaving enough room to null terminate the string. The loop will read characters until the end of the line, end of file, or an error has occurred. |
| #include <stdio.h>    enum { BUFFERSIZE = 32 };    void func(void) {  char buf[BUFFERSIZE];  int ch;  size\_t index = 0;  bool truncated = false;    while ((ch = getchar()) != '\n' && ch != EOF) {  if (index < sizeof(buf) - 1) {  buf[index++] = (char)ch;  } else {  truncated = true;  }  }  buf[index] = '\0'; /\* Terminate string \*/  if (ch == EOF) {  /\* Handle EOF or error \*/  }  if (truncated) {  /\* Handle truncation \*/  }  } |
| In this solution the call to fscanf() is constricted to not overflow buf. |
| #include <stdio.h>    enum { BUF\_LENGTH = 1024 };    void get\_data(void) {  char buf[BUF\_LENGTH];  if (1 != fscanf(stdin, "%1023s", buf)) {  /\* Handle error \*/  }    /\* Rest of function \*/  } |
| The strlen() function can be used to determine the length of the strings referenced by argv[0] through argv[arc – 1] so that adequate memory can be dynamically allocated. |
| #include <stdlib.h>  #include <string.h>    int main(int argc, char \*argv[]) {  /\* Ensure argv[0] is not null \*/  const char \*const name = (argc && argv[0]) ? argv[0] : "";  char \*prog\_name = (char \*)malloc(strlen(name) + 1);  if (prog\_name != NULL) {  strcpy(prog\_name, name);  } else {  /\* Handle error \*/  }  free(prog\_name);  return 0;  } |
| The strcpy\_s() function provides additional safeguards, including accepting the size of the destination buffer as an additional argument. |
| #define \_\_STDC\_WANT\_LIB\_EXT1\_\_ 1  #include <stdlib.h>  #include <string.h>    int main(int argc, char \*argv[]) {  /\* Ensure argv[0] is not null \*/  const char \*const name = (argc && argv[0]) ? argv[0] : "";  char \*prog\_name;  size\_t prog\_size;    prog\_size = strlen(name) + 1;  prog\_name = (char \*)malloc(prog\_size);    if (prog\_name != NULL) {  if (strcpy\_s(prog\_name, prog\_size, name)) {  /\* Handle error \*/  }  } else {  /\* Handle error \*/  }  /\* ... \*/  free(prog\_name);  return 0;  } |
| Environmental variables are loaded into memory when the program is loaded. As a result, the length of the strings can be determined by calling strlen(), and the resulting length can be used to allocate the dynamic memory. |
| #include <stdlib.h>  #include <string.h>    void func(void) {  char \*buff;  char \*editor = getenv("EDITOR");  if (editor == NULL) {  /\* EDITOR environment variable not set \*/  } else {  size\_t len = strlen(editor) + 1;  buff = (char \*)malloc(len);  if (buff == NULL) {  /\* Handle error \*/  }  memcpy(buff, editor, len);  free(buff);  }  } |
| If the precision is specified, no more than that many bytes are written. The precision 123 in this solution ensures that filename can contain the first 123 characters of name, the .txt extension, and null terminator. |
| #include <stdio.h>    void func(const char \*name) {  char filename[128];  sprintf(filename, "%.123s.txt", name);  } |
| A more general solution is to use snprintf() function, which also truncates name if it will not fit within the filename. |
| #include <stdio.h>    void func(const char \*name) {  char filename[128];  int result = snprintf(filename, sizeof(filename), "%s.txt", name);  if (result != strlen(filename) {  /\* truncation occurred \*/  }  } |

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

| **Principles(s):**  Validate input data – this vulnerability maps to this principle because we need to validate the input data to be saved and it cannot be truncated if it is being saved within a database.  Sanitize data sent to other systems – it also needs to be sanitized if the data is being sent to other systems to protect it against malware or a malicious attack. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | P18 | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | N/A | Astrée reports all buffer overflows resulting from copying data to a buffer that is not large enough to hold that data. |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR31 | Detects calls to unsafe string function that may cause buffer overflow  Detects potential buffer overruns, including those caused by unsafe usage of fscanf() |
| CodeSonar | 8.1p0 | LANG.MEM.BO  LANG.MEM.TO  MISC.MEM.NTERM  BADFUNC.BO.\* | Buffer overrun  Type overrun  No space for null terminator  A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Compass/ROSE | N/A | N/A | Can detect violations of the rule. However, it is unable to handle cases involving strcpy\_s() or manual string copies such as the one in the first example |
| Coverity | 2017.07 | STRING\_OBERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| Fortify SCA | 5.0 | N/A | N/A |
| Helix QAC | 2024.1 | C2840, C5009, C5038  C++0145, C++5009, C++5038  DF2840, DF2841, DF2842, DF2843, DF2845, DF2846, DF2847, DF2848, DF2930, DF2931, DF2932, DF2933, DF2935, DF2936, DF2937, DF2938 | N/A |
| Klocwork | 2024.1 | C2840, C5009, C5038  C++0145, C++5009, C++5038  DF2840, DF2841, DF2842, DF2843, DF2845, DF2846, DF2847, DF2848, DF2930, DF2931, DF2932, DF2933, DF2935, DF2936, DF2937, DF2938 | N/A |
| LDRA tool suite | 9.7.1 | 489 s, 109 D, 66X, 70 X, 71X | Partially implemented |
| Parasoft  C/C++test | 2023.1 | CERT\_C-STR31-a  CERT\_C-STR31-b  CERT\_C-STR31-c  CERT\_C-STR31-d  CERT\_C-STR31-e | Avoid accessing arrays out of bounds  Avoid overflow when writing to a buffer  Prevent buffer overflows from tainted data  Avoid buffer write overflow from tainted data  Avoid using unsafe string functions which may cause buffer overflows |
| PC-Init Plus | 1.4 | 421, 498 | Partially supported |
| Polyspace Big Finder | R2024a | CERT C:Rule STR31-C | 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  Insufficient destination buffer size  Rule partially covered. |
| PVS-Studio | 7.31 | V518, V645, V 727V V755 | N/A |
| Splint | 3.1.1 | N/A | N/A |
| TrustInSoft Analyzer | 1.38 | Mem\_access | Exhaustively verified |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-ERR34-C | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| This example converts the string token stored within the buff to a signed integer value using the atoi() function. |
| #include <stdlib.h>    void func(const char \*buff) {  int si;    if (buff) {  si = atoi(buff);  } else {  /\* Handle error \*/  }  } |
| This example uses the sscanf() function to convert a string token to an integer. The sscanf() function has the same limitations as atoi(). |
| #include <stdio.h>    void func(const char \*buff) {  int matches;  int si;    if (buff) {  matches = sscanf(buff, "%d", &si);  if (matches != 1) {  /\* Handle error \*/  }  } else {  /\* Handle error \*/  }  } |

| **Compliant Code** |
| --- |
| This solution uses strtol() to convert a string token to an integer and ensures the value is within range of 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) {  (void) fprintf(stderr, "%s: not a decimal number\n", buff);  } else if ('\0' != \*end) {  (void) fprintf(stderr, "%s: extra characters at end of input: %s\n", buff, end);  } else if ((LONG\_MIN == sl || LONG\_MAX == sl) && ERANGE == errno) {  (void) fprintf(stderr, "%s out of range of type long\n", buff);  } else if (sl > INT\_MAX) {  (void) fprintf(stderr, "%ld greater than INT\_MAX\n", sl);  } else if (sl < INT\_MIN) {  (void) 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):** Adopt a secure coding standard – This vulnerability maps to this principle because secure coding techniques are a must for the target language. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ERR34 | N/A |
| Clang | 3.9 | Cert-err34-c | Checked by clang-tidy |
| CodeSonar | 8.1p0 | BADFUNC.ATOF  BADFUNC.ATOI  BADFUNC.ATOL  BADFUNC.ATOLL  (customization) | Use of atof  Use of atoi  Use of atol  Use of atoll  Users can add custom checks for uses of other undesirable conversion functions. |
| Compass/ROSE | N/A | N/A | Can detect violations of this recommendation by flagging invocations of the following functions:  atoi()  scanf(),  fscanf(),  sscanf() |
| HelixQAC | 2024.1 | C5030  C++5016 | N/A |
| Klockwork | 2024.1 | CERT.ERR.CONV.STR\_TO\_NUM  MISRA.STDLIB.ATOI  SV.BANNED.RECOMMENDED.SCANF | N/A |
| LDRA tool suite | 9.7.1 | 44 S | Fully implemented |
| Parasoft  C/C++test | 2023.1 | CERT\_C-ERR34-a | The library functions atof, atoi and atoll from library stdlib.h shall not be used |
| PC-Init Plus | 1.4 | 586 | Assistance provided |
| Polyspace Big Finder | R2024a | CERT C: Rule ERR34-C | Checks for unsafe conversion from string to numeric value (rule fully covered) |
| SonarQube  C/C++ Plugin | 3.11 | S989 | N/A |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-MEM31-C | Free Dynamically allocated memory when no longer needed |

| **Noncompliant Code** |
| --- |
| The object allocated by the call to mallock() isn’t freed before the end of lifecycle of the last pointer text\_buffer referencing the object. |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }  return 0;  } |

| **Compliant Code** |
| --- |
| The pointer is deallocated with a call to freee() |
| #include <stdlib.h>    enum { BUFFER\_SIZE = 32 };    int f(void) {  char \*text\_buffer = (char \*)malloc(BUFFER\_SIZE);  if (text\_buffer == NULL) {  return -1;  }    free(text\_buffer);  return 0;  } |

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

| **Principles(s):** Use effective quality assurance techniques – this vulnerability maps to this principle because if the memory is not freed then the overall quality of the code is reduced. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Probable | Medium | P8 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 24.04 | N/A | Supported, but no explicit checker |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM31 | Can detect dynamically allocated resources that are not freed |
| CodeSonar | 8.1p0 | ALLOC.LEAK | Leak |
| Compass/ROSE | N/A | N/A | N/A |
| Coverity | 2017.07 | RESOURCE\_LEAK  ALLOC\_FREE\_MISMATCH | Finds resource leaks from variables that go out of scope while owning a resource |
| Cppcheck | 1.66 | LeakReturnValNotUsed | Does not use return value of memory allocation function |
| Helix QAC | 2024.1 | DF2706, DF2707, DF2708  C++3337, C++ 3338 | N/A |
| Klocwork | 2024.1 | CL.FFM.ASSIGN  CL.FFM.COPY  CL.SHALLOW.ASSIGN  CL.SHALLOW.COPY  FMM.MIGHT  FMM.MUST | N/A |
| LDRA tool suite | 9.7.1 | 50 D | Partially implemented |
| Parasoft C/C++test | 2023.1 | CERT\_CMEM31-a | Ensure resources are freed |
| Parasoft Insure ++ | N/A | N/A | Runtime analysis |
| Polyspace Bug Finder | R2024a | CERT C:Rule MEM31-C | Checks for memory leak (rule fully covered) |
| PVE-Studio | 7.31 | V773 | N/A |
| SonarQube C/C++ Plugin | 3.11 | S3584 | N/A |
| Splint | 3.1.1 | N/A | N/A |
| TrustInSoft analyzer | 1.38 | Malloc | Exhaustively verified |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-ERR32-C | Do not rely on indeterminate values of errno |

| **Noncompliant Code** |
| --- |
| The handler function in this example attempts to restore handling for the signal indicated by signum. If the request to set the signal to default can be done, the signal function returns the value of the signal handler for the most recent successful call to the signal function or specified signal |
| #include <signal.h>  #include <stdlib.h>  #include <stdio.h>    typedef void (\*pfv)(int);    void handler(int signum) {  pfv old\_handler = signal(signum, SIG\_DFL);  if (old\_handler == SIG\_ERR) {  perror("SIGINT handler"); /\* Undefined behavior \*/  /\* Handle error \*/  }  }    int main(void) {  pfv old\_handler = signal(SIGINT, handler);  if (old\_handler == SIG\_ERR) {  perror("SIGINT handler");  /\* Handle error \*/  }    /\* Main code loop \*/    return EXIT\_SUCCESS;  } |
| The signal in this code example alters the value of errno. As a result, it can cause incorrect handling if executed between a failed function call and the subsequent inspection of errno |
| #include <signal.h>  #include <stdlib.h>  #include <errno.h>  #include <sys/wait.h>    void reaper(int signum) {  errno = 0;  for (;;) {  int rc = waitpid(-1, NULL, WNOHANG);  if ((0 == rc) || (-1 == rc && EINTR != errno)) {  break;  }  }  if (ECHILD != errno) {  /\* Handle error \*/  }  }    int main(void) {  struct sigaction act;  act.sa\_handler = reaper;  act.sa\_flags = 0;  if (sigemptyset(&act.sa\_mask) != 0) {  /\* Handle error \*/  }  if (sigaction(SIGCHLD, &act, NULL) != 0) {  /\* Handle error \*/  }    /\* ... \*/    return EXIT\_SUCCESS;  } |

| **Compliant Code** |
| --- |
| This solution does not reference errno and doesn’t return from the signal handler if the signal() call fails |
| #include <signal.h>  #include <stdlib.h>  #include <stdio.h>    typedef void (\*pfv)(int);    void handler(int signum) {  pfv old\_handler = signal(signum, SIG\_DFL);  if (old\_handler == SIG\_ERR) {  abort();  }  }    int main(void) {  pfv old\_handler = signal(SIGINT, handler);  if (old\_handler == SIG\_ERR) {  perror("SIGINT handler");  /\* Handle error \*/  }    /\* Main code loop \*/    return EXIT\_SUCCESS;  } |
| This solution saves and restores the value of errno in the signal handler |
| #include <signal.h>  #include <stdlib.h>  #include <errno.h>  #include <sys/wait.h>    void reaper(int signum) {  errno\_t save\_errno = errno;  errno = 0;  for (;;) {  int rc = waitpid(-1, NULL, WNOHANG);  if ((0 == rc) || (-1 == rc && EINTR != errno)) {  break;  }  }  if (ECHILD != errno) {  /\* Handle error \*/  }  errno = save\_errno;  }    int main(void) {  struct sigaction act;  act.sa\_handler = reaper;  act.sa\_flags = 0;  if (sigemptyset(&act.sa\_mask) != 0) {  /\* Handle error \*/  }  if (sigaction(SIGCHLD, &act, NULL) != 0) {  /\* Handle error \*/  }    /\* ... \*/    return EXIT\_SUCCESS;  } |

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

| **Principles(s):** Practice defense in depth – this vulnerability maps to this principle because if the program throws an abnormal behavior, then it could leave the application vulnerable to injection attacks.  Default Deny- This vulnerability maps to this principle because errno is not returned from the signal handler. Meaning that if errno is altered, then the system will not return it and close entirely. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | P3 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ERR32 | N/A |
| Compass/ROSE | N/A | N/A | Could detect violations of this rule by looking for signal handlers that themselves call signal(). A violation is reported if the call fails and the handler therefore checks errno. A violation also exists if the signal handler modifies errno without first copying its value elsewhere |
| Coverity | 2017.07 | MISRA C 2012 Rule 22.8  MISRA C 2012 Rule 22.9  MISRA C 2012 Rule 22.10 | Implemented |
| HelixQAC | 2024.1 | C2031  DF4781, DF4782, DF4783 | N/A |
| Klockwork | 2024.1 | MISRA.INCL.SIGNAL.2012  MISRA.STDLIB.SIGNAL | N/A |
| LDRA tool suite | 9.7.1 | 44 S | Enhanced Enforcement |
| Parasoft  C/C++test | 2023.1 | CERT\_C-ERR32-a | Properly use errno value |
| Polyspace Bug Finder | R2024a | CERT C: Rule ERR32-C | Checks for misuse of errno in a signal handler (rule fully covered) |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-ERR51-CPP | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In this example, neither f() nor main() catch exceptions thrown by throwing\_func(). Because no matching handler can be found for the exception thrown, std::terminate() is called |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  f();  } |
| In this example, the thread entry point function thread\_start() doesn’t catch exceptions thrown by throwing\_func(). If the initial thread function exists because an exception is thrown std::terminate() is called |
| #include <thread>    void throwing\_func() noexcept(false);    void thread\_start() {  throwing\_func();  }    void f() {  std::thread t(thread\_start);  t.join();  } |

| **Compliant Code** |
| --- |
| With this example, the main entry point handles all exceptions, which ensures that the stack is unwound up to main() function and allows for graceful management of external resources. |
| void throwing\_func() noexcept(false);    void f() {  throwing\_func();  }    int main() {  try {  f();  } catch (...) {  // Handle error  }  } |
| In this solution, the thread\_start() handles all exceptions and doesn’t rethrow, allowing the thread to terminate normally. |
| #include <thread>    void throwing\_func() noexcept(false);    void thread\_start(void) {  try {  throwing\_func();  } catch (...) {  // Handle error  }  }    void f() {  std::thread t(thread\_start);  t.join();  } |

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

| **Principles(s):**  Architect and design for security policies- this vulnerability maps to this principle because if the system acts a different way than anticipated, then the design should be implemented for it to not allow the unanticipated behavior to reveal any vulnerable data.  Head compiler warnings – the program can use assertions to check for exception handling and making sure that the program handles the exceptions accordingly. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 | N/A |
| CodeSonar | 8.1p0 | LANG.STRUCT.UCTCH | Unreachable Catch |
| Helix QAC | 2024.1 | C++4035, C++ 4036, C++4037 | N/A |
| Klocwork | 2024.1 | MISRA.CATCH.ALL | N/A |
| LDRA tool suite | 9.7.1 | 527 S | Partially Implemented |
| Parasoft  C/C++test | 2023.1 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| Polyspace Bug Finder | R2024a | CERT C++:ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| RuleChecker | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Handling | STD-ERR52-CPP | Do not use setjmp() or longjmp() |

| **Noncompliant Code** |
| --- |
| In this example, the call to longjmp() occurs in a context with a local Counter object. Since this objects destructor is nontrivial, undefined behavior results |
| #include <csetjmp>  #include <iostream>    static jmp\_buf env;    struct Counter {  static int instances;  Counter() { ++instances; }  ~Counter() { --instances; }  };    int Counter::instances = 0;    void f() {  Counter c;  std::cout << "f(): Instances: " << Counter::instances << std::endl;  std::longjmp(env, 1);  }    int main() {  std::cout << "Before setjmp(): Instances: " << Counter::instances << std::endl;  if (setjmp(env) == 0) {  f();  } else {  std::cout << "From longjmp(): Instances: " << Counter::instances << std::endl;  }  std::cout << "After longjmp(): Instances: " << Counter::instances << std::endl;  } |

| **Compliant Code** |
| --- |
| This solution replaces the calls to setjmp() and longjmp() with a throw expression and catch statement |
| #include <iostream>    struct Counter {  static int instances;  Counter() { ++instances; }  ~Counter() { --instances; }  };    int Counter::instances = 0;    void f() {  Counter c;  std::cout << "f(): Instances: " << Counter::instances << std::endl;  throw "Exception";  }    int main() {  std::cout << "Before throw: Instances: " << Counter::instances << std::endl;  try {  f();  } catch (const char \*E) {  std::cout << "From catch: Instances: " << Counter::instances << std::endl;  }  std::cout << "After catch: Instances: " << Counter::instances << std::endl;  } |

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

| **Principles(s):** Keep it simple – this vulnerability maps to this principle because you should implement try, catch methods rather than fancy methods that might cause the system to be vulnerable. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Probable | Medium | P4 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Include-setjmp | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR52 | N/A |
| Clang | 3.9 | Cert-err52-cpp | Checked by clang-tidy |
| Helix QAC | 2024.1 | C++5015 | N/A |
| Klockwork | 2024.1 | MISRA.STDLIB.LONGJMP | N/A |
| LDRA tool suite | 9.7.1 | 43 S | Fully implemented |
| Parasoft  C/C++test | 2023.1 | CER\_CPP-ERR52-a  CER\_CPP-ERR52-b | The facilities provided by <setjmp.h> should not be used  The standard header files <setjmp.h> or <csetjmp> shall not be used |
| Polyspace Bug Finder | R2024a | CERT C++ ERR52-CPP | Checks for use of setjmp/longjmp (rule fully covered) |
| RuleChekcer | 22.10 | Include-setjmp | Fully checked |
| SonarQube  C/C++ Plugin | 4.10 | S982 | N/A |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Input Output (I/O) | STD- FIO50-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 from the same file. Since there is no intervening positioning call between the formatted 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** |
| --- |
| In this solution, the std::basic\_istream<T>::seekg() function is called between the output and input, eliminating 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 - this vulnerability maps to this principle because if the user types in something that is not validated, it could lead to an injection attack, and they could get sensitive data that is being stored.  Adhere to the Principle of least Privilege – this vulnerability maps to this principle because it is only opening the file rather than giving permission to edit the contents. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P6 | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-FiO50 | N/A |
| CodeSonar | 8.1p0 | IO.IOWOP  IO.OIWOP | Input after output without positioning  Output after input without positioning |
| Helix QAC | 2024.1 | DF4711, DF4712, DF4713 | N/A |
| Parasoft  C/C++test | 2023.1 | CERT\_CPP-FIO50-a | Do not alternately input and output from a stream without an intervening flush or positioning call |
| Polyspace Bug Fidner | R2024a | CERT C++: FIO50-CPP | Checks for alternating input and output from a stream without flush or positioning call (rule fully covered) |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Integers(INT) | STD-INT50-CPP | Do not cast to an out-of-range enumeration value |

| **Noncompliant Code** |
| --- |
| This example attempts to check whether a given value is within the range of acceptable enumeration values. It is doing it after casting to the enumeration type, which may not be able to represent the given integer value. |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);    if (enumVar < First || enumVar > Third) {  // Handle error  }  } |

| **Compliant Code** |
| --- |
| This solution checks that the value can represented by the enumeration type before performing the conversion to guarantee the conversion doesn’t result in an unspecified type. It does this by restricting the converted value to one for which there is a specific enumerator value |
| enum EnumType {  First,  Second,  Third  };    void f(int intVar) {  if (intVar < First || intVar > Third) {  // Handle error  }  EnumType enumVar = static\_cast<EnumType>(intVar);  } |
| This solution uses a scoped enumeration, which has a fixed underlying int type by default, allowing any value from the parameter to be converted into a valid enumeration value. IT doesn’t restrict the converted value to one for which there is a specific enumerator value |
| enum class EnumType {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  } |
| This solution uses an unscoped enumeration but provides a fixed underlying int type allowing any value from the parameter to be converted to a valid enumeration value |
| enum EnumType : int {  First,  Second,  Third  };    void f(int intVar) {  EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):**  Validate input data – this vulnerability maps to this principle because with a cast to an out-of-range enumeration value it will cause an exception to be thrown.  Practice defense in depth – this vulnerability also maps to this principle because it can lead to an injection attack if an out-of-range cast is being called. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Medium | P4 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 22.10 | Cast-integer-to-enum | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-INT50 | N/A |
| CodeSonar | 8.1p0 | LANG.CAST,COERCE  LANG.CAST.VALUE | Corecion alters value  Cast alters value |
| Helix QAC | 2024.1 | C++3013 | N/A |
| Parasoft  C/C++test | 2023.1 | CERT\_CPP-INT50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PVS-Studio | 7.31 | V1016 | N/A |
| RuleChecker | 22.10 | Cast-integer-to-enum | Partially checked |
| Polyspace Bug Finder | R2024a | CERT C++: INT50-CPP | Checks for casting to out-of range enumeration value (rule fully 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.

When looking at the DevOps process to automate enforcement of the standards in this policy I would implement a couple of automated tests within the Build section of the Pre-production section of the diagram above. This would eliminate shrink the Verify and test section, and it will help reach the Transition and health check section faster within the DevOps diagram. Using automated tools will help developers during the build process to catch mistakes early and fix them earlier in the development cycle.

Implementing automated tools such as Static code analysis is essential to the structural as well as the composition of the overall program being built. Catching mistakes early in the development process will help reduce cost as well as the overall time to release the application for public use. Also testing early and frequently will help the company’s reputation of being secure ang gain trust of the users who are going to be utilizing the program being built.

### 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-FLP30-C | Low | Probable | Low | P6 | 2 |
| STD-MEM31-C | Medium | Unlikely | Medium | P8 | 2 |
| STD-STR31-C | High | Likely | Medium | P18 | 1 |
| STD-ERR32-C | Low | Unlikely | Low | P3 | 3 |
| STD-ERR34-C | Medium | Unlikely | Medium | P4 | 3 |
| STD-INT35-C | Low | Unlikely | Medium | P2 | 3 |
| STD-FIO50-CPP | Low | Unlikely | Medium | P6 | 2 |
| STD-INT50-CPP | Medium | Unlikely | Medium | P4 | 3 |
| STD-ERR51-CPP | Low | Probable | Medium | P4 | 3 |
| STD-ERR52-CPP | Low | Probable | Medium | P4 | 3 |

### Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption at rest | Is the practice of protecting stored data by encoding it using encryption algorithms. This policy applies because even when the data in the database is not being used, it needs to be always protected. In order to decipher the encryption, you need a key. The key itself ensures the safety of the confidential data. |
| Encryption in flight | This is a technique that is used to keep the privacy of communication data when it travels between two points within the system. This policy applies because the data needs to be protected when it is being communicated between systematic points. This also protects the data if someone was to intervene and intercept the data before it got to the correct place within the system. |
| Encryption in use | This itself is a cybersecurity practice of encrypting the data as it is being actively accessed and/or processed. This policy applies to encryption in use because if the data is being accessed or processed, then it needs to be encrypted to prevent exposure from attackers, as well as theft of the data that is being accessed. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This is the first step in the Triple A framework. This identifies the user and ensures whom they claim to be. This policy applies because in order to access the client information, the user must be able to identify themselves and prove who they are in order to access the system |
| Authorization | This portion of the Triple A framework enforces the policies and procedures, such as, the users authorization levels. This policy applies because in order for the user to access the information of another, they need to have administration that allows them to do so. If the current user adds new users to the program, then they must have access to do so. This is extremely important when it comes to security of the data. |
| Accounting | This measures the resources the user consumes during their time within the system. It basically logs the data that the user accessed when they were utilizing the application. This is important for accountability reasons, for example, if something goes missing, then the administration staff can see who the last person to access the data would be and what they did with it. |

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

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

### Map the Principles

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

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

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

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

## Audit Controls and Management

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

Evidence will include the following:

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

## Enforcement

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

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

## Exceptions Process

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

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

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

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

## Distribution

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

## Policy Change Control

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

## Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 05/25/2024 | Update to the standards and | Robert Lowrey | [Insert text.] |
| 1.2 | 6/15/2024 | Updated Security Policy | Robert Lowrey | [Insert text.] |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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