**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 | Another name is data validation, is the proper testing of any input supplied by a user application. It prevents improperly formed data from entering an information system. This is because it is difficult to detect malicious user who is trying to attack software, applications should check and validate all input entered into system. Input validation should occur when data is received from an external party, especially if the data is from untrusted sources. Incorrect input validation can lead to injection attacks, memory leakage, and compromised systems. |
| 1. Heed Compiler Warnings | They comply code using the highest warning level available for your complier and eliminates warnings by modifying the code. The complier’s designers build in the warnings to help catch bugs before testing, which means the cost to fix the problem is reduced. Warnings catch bugs that are hard to find in testing, which means the cost to fix the problem is reduced. And if you maintain a warning- free codebase, and warnings all of sudden start appearing, it is easy to pinpoint where the problem is coming from. |
| 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. It also looks at how information security controls and safeguards are implemented in IT systems in order to protect the confidentiality, integrity, and availability of the data that are used, processed and stored in those systems. |
| 1. Keep It Simple | 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. |
| 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. Basically, this means unless you specifically allow something, you deny it. It’s the network version of whitelisting. In you perimeter device (most likely a firewall),you define the ports and protocols you allow, and turn everything else off. |
| 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. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data passed to complex subsystems such command shells, relational databases, and commercial off the shelf components. Attackers may be able to invoke unused functionality in these components through the use of SQL, command, or other injection attacks. This 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. |
| 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 of a 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. |
| 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, identify and correcting invalid assumptions. |
| 1. Adopt a Secure Coding Standard | Develop and or apply a secure coding standard for your target development language platform. They are also practicing that are implemented to prevent the introduction of security vulnerabilities, such as bugs and logic laws. By following secure coding standards, companies can significantly reduce vulnerabilities before deployment. |

## 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] | Validate Input Data |

| **Noncompliant Code** |
| --- |
| No matter how many times this code is executed, it will always produce the same sequence. |
| #include <random>  #include <iostream>  Void f() {  Std::mt19937 engine;  For(int i = 0; I < 10; ++i){  Std::cout << engine() << “ , “,;  }  } |

| **Compliant Code** |
| --- |
| This device generates random outputs. |
| #include <random>  #include <iostream>  Void f() {  Std::random\_device dev;  Std::mt19937 enegine(dev());  For(int i = 0 ; i < 10; ++i){  Std:: cout<< engine() << “, “,;  }  } |

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

|  |
| --- |
| **Principles(s):** Validate input data, because it prevents improper data inputing. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | probable | medium | P4 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.4.1 | Static code analysis | a [static analysis tool](http://en.wikipedia.org/wiki/Static_analysis_tool) for C/C++ code. It provides [unique code analysis](http://cppcheck.sourceforge.net/#unique) to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs. The goal is to have very few false positives. Cppcheck is designed to be able to analyze your C/C++ code even if it has non-standard syntax (common in embedded projects). |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | Validate Data Value |

| **Noncompliant Code** |
| --- |
| Type range errors, including loss of data(truncation) and loss of sign (sign errors), can occur when converting from a value of an unsigned integer type to a value of a signed integer type. |
| #include <limits.h>  Void func(void) {  unsigned long int u\_a = ULONG\_MAX;  signed char sc;  sc = (signed char)u\_a;/\* Cast eliminates warning \*/  /\* …\*/  } |
|  |

| **Compliant Code** |
| --- |
| Validate ranges when converting from a signed type to an unsigned type. |
| #include<limits.h>  Void func(signed int si){  Unsigned int ui;  if(si <0) {  /\*Handle error\*/  } else {  Ui = (unsigned int) si; /\*Cast eliminates warning\*/  }  /\*…\*/  }  /\*..\*/  Func(INT\_MIN +1); |

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

|  |
| --- |
| **Principles(s):** Validate input data, because it is validating each range when it is converting and making sure it is input properly. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Compass/ROSE |  | [Insert text.] | Can detect violations of this rule. |
| Coverity | 2017.07 | Negative\_Returms  Reverse\_negative  Misra\_cast | Can find array accesses, loop bounds and other expressions that may contain dangerous implied integer conversions that would result in unexpected behavior |
| CPPcheck | 1.66 | memsetValueOutOfRange | The second argument to memset()cannot be represented as unsigned char |
| Parasoft C/C++ test | 2020.2 | CERT\_C-INT31-a  CERT\_C-INT31-b  CERT\_C-INT31-c  CERT\_C-INT31-d  CERT\_C-INT31-e  CERT\_C-INT31-f  CERT\_C-INT31-g  CERT\_C-INT31-h  CERT\_C-INT31-i  CERT\_C-INT31-j  CERT\_C-INT31-k  CERT\_C-INT31-l  CERT\_C-INT31-m  CERT\_C-INT31-n  CERT\_C-INT31-o | An expression of essentially Boolean type should always be used where an operand is interpreted as a Boolean value.  An operand of essentially Boolean type should not be used where an operand is interpreted as a numeric value An operand of essentially character type should not be used where an operand is interpreted as a numeric value An operand of essentially enum type should not be used in an arithmetic operation Shift and bitwise operations should not be performed on operands of essentially signed or enum type An operand of essentially signed or enum type should not be used as the right hand operand to the bitwise shifting operator An operand of essentially unsigned type should not be used as the operand to the unary minus operator The value of an expression shall not be assigned to an object with a narrower essential type The value of an expression shall not be assigned to an object of a different essential type category Both operands of an operator in which the usual arithmetic conversions are performed shall have the same essential type category The second and third operands of the ternary operator shall have the same essential type category The value of a composite expression shall not be assigned to an object with wider essential type If a composite expression is used as one operand of an operator in which the usual arithmetic conversions are performed then the other operand shall not have wider essential type If a composite expression is used as one (second or third) operand of a conditional operator then the other operand shall not have wider essential type Avoid integer overflows |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | Data Type validation |

| **Noncompliant Code** |
| --- |
| In a hosted environment, arguments read from the command line are stored in process memory. The function main(), called at program start up, is typically declared as follows when the program accepts command line arguments |
| Int main(int argc, char \*argv[]){/\* …\*/} |

| **Compliant Code** |
| --- |
| The call to fscanf() is constrained not to overflow buf |
| Enum{ BUF\_LENGTH = 1024};  Void get\_data(void){  Char buf[BUF\_LENGTH];  if (1 != fscanf(stdin, “%1023s”, buf)){  \*/Handle error\*/  }  /\*Rest of Function\*/  } |

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

|  |
| --- |
| **Principles(s):** Validate input data, because it validates the string |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | CertC-STR31 | Detects calls to unsafe string function that may cause buffer overflow. Detects potential buffer overruns, including those caused by unsafe usage of fscanf() |
| CodeSonar | 6.0p0 | LANG.MEM.BO  LANG.MEM.TO  MISC.MEM.NTERM  BADFUNC.BO\* | Buffer overrun  Type overrun  No space for null terminator  A collection of warning classes that report uses of library functions prone to internal buffer overflows |
| Coverity | 2017.07 | STRING\_OVERFLOW  BUFFER\_SIZE  OVERRUN  STRING\_SIZE | Fully implemented |
| LDRA tool suite | 9.7.1 | 489 S, 109 D, 66 X, 70 X, 71 X | Partially implemented |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | SQL Injection |

| **Noncompliant Code** |
| --- |
| The incorrect\_password() function in this noncompliant code example is called during identification and authentication to display an error message if the specified user is not found or the password is incorrect. The function accepts the name of the user as a string referenced by user. This is an exemplar of untrusted data that originates from an unauthenticated user. |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    void incorrect\_password(const **char** \*user) {  **int** ret;    /\* User names are restricted to 256 or fewer characters \*/    static const **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + sizeof(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);    if (msg == NULL) {      /\* Handle error \*/    }    ret = snprintf(msg, len, msg\_format, user);    if (ret < 0) {      /\* Handle error \*/    } else if (ret >= len) {      /\* Handle truncated output \*/    }  **fprintf**(stderr, msg);  **free**(msg);  } |

| **Compliant Code** |
| --- |
| This compliant solution fixes the problem by replacing the fprintf() call with a call to fputs(), which outputs msg directly to stderr without evaluating its contents |
| #include <stdio.h>  #include <stdlib.h>  #include <string.h>    void incorrect\_password(const **char** \*user) {  **int** ret;    /\* User names are restricted to 256 or fewer characters \*/    static const **char** msg\_format[] = "%s cannot be authenticated.\n";  **size\_t** len = **strlen**(user) + sizeof(msg\_format);  **char** \*msg = (**char** \*)**malloc**(len);    if (msg == NULL) {      /\* Handle error \*/    }    ret = snprintf(msg, len, msg\_format, user);    if (ret < 0) {      /\* Handle error \*/    } else if (ret >= len) {      /\* Handle truncated output \*/    }  **fputs**(msg, stderr);  **free**(msg);  } |

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

|  |
| --- |
| **Principles(s):** Validate input data and sanitize data sent to other systems because it has to stop ability to invoke side functions while validating the data. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | [Insert text.] | Supported via stubbing/taint analysis |
| Axivion Bauhaus Suite | 6.9.0 | IO.INJ>FMT  MISC.FMT | Format strings injection  Format string |
| Coverity | 2017.07 | TAINTED\_STRING | Implemented |
| Klockwork | 2018 | SV.FMTSTR.GENERIC  SV.TAINTED.FMTSTR | [Insert text.] |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | Control Memory Access Rights |

| **Noncompliant Code** |
| --- |
| In this noncompliant example, the object allocated by the call to malloc() is not freed before the end of the lifetime of the last pointer text buffer referring to 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** |
| --- |
| In this complaint solution, the pointer is deallocated with a call to free() |
| #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):** Architect and Design for Security Policies because it controls memory access rights or privilege on a computer or software. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | [Insert text.] | Supported, but no explicit checker |
| TrustInSoft Analyzer | 1.38 | Malloc | Exhaustively verified |
| SonarQube C/C++ Plugin | 3.11 | S3584 | [Insert text.] |
| Polyspace Bug Finder | R2021a | Cert C: Rule MEM31-C | Checks for memory leak(rule fully covered) |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-060-CPP] | Testing |

| **Noncompliant Code** |
| --- |
| This noncompliant code example converts the string token stored in 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 \*/    }  } |

| **Compliant Code** |
| --- |
| The strtol(), strtoll(), strtoimax()), strtoul(), strtoull(), strtoumax(), strtof(), strtod(), and strtold() functions convert a null-terminated byte string to long int, long long int, intmax\_t, unsigned long int, unsigned long long int, uintmax\_t, float, double, and long double representation, respectively.  This compliant solution uses strtol() to convert a string token to an integer and ensures that the value is in the 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) {  **fprintf**(stderr, "%s: not a decimal number\n", buff);    } else if ('\0' != \*end) {  **fprintf**(stderr, "%s: extra characters at end of input: %s\n", buff, end);    } else if ((LONG\_MIN == sl || LONG\_MAX == sl) && ERANGE == **errno**) {  **fprintf**(stderr, "%s out of range of type long\n", buff);    } else if (sl > INT\_MAX) {  **fprintf**(stderr, "%ld greater than INT\_MAX\n", sl);    } else if (sl < INT\_MIN) {  **fprintf**(stderr, "%ld less than INT\_MIN\n", sl);    } else {      si = (**int**)sl;        /\* Process si \*/    }  } |

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

|  |
| --- |
| **Principles(s):** Validate input data, Use Effective Quality Assurance Techniques because it is testing the code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 6.9.0 | Csrtc-err34 | [Insert text.] |
| Clang | 3.9 | Cert-err34-c | Checked by clang-tidy |
| CodeSonar | 6.0p0 | BADFUNC.ATOF  BADFUNC.ATOI  BADFUNC.ATOL  BADFUNC.ATOLL  (customization) | Use of atof  Use of atoi  Use of atoll  Use of atoll  Users can add custom checks for users of other undesirable conversion functions |
| Compass/ROSE | [Insert text.] | [Insert text.] | Can detect violations of this recommendation by flagging invocations of the following functions:  Atoi()  Scanf(), fscanf(), sscanf()  others |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exception handling |

| **Noncompliant Code** |
| --- |
| In this noncompliant code example, temp\_num, tmp2, and num\_of\_records are derived from a tainted source. Consequently, an attacker can easily cause calloc() to fail by providing a large value for num\_of\_records. |
| #include <stdlib.h>  #include <string.h>    enum { SIG\_DESC\_SIZE = 32 };    typedef struct {  **char** sig\_desc[SIG\_DESC\_SIZE];  } signal\_info;    void func(**size\_t** num\_of\_records, **size\_t** temp\_num,            const **char** \*tmp2, **size\_t** tmp2\_size\_bytes) {    signal\_info \*start = (signal\_info \*)**calloc**(num\_of\_records,                                            sizeof(signal\_info));      if (tmp2 == NULL) {      /\* Handle error \*/    } else if (temp\_num > num\_of\_records) {      /\* Handle error \*/    } else if (tmp2\_size\_bytes < SIG\_DESC\_SIZE) {      /\* Handle error \*/    }      signal\_info \*point = start + temp\_num - 1;  **memcpy**(point->sig\_desc, tmp2, SIG\_DESC\_SIZE);    point->sig\_desc[SIG\_DESC\_SIZE - 1] = '\0';    /\* ... \*/  **free**(start);  } |

| **Compliant Code** |
| --- |
| To correct this error, ensure the pointer returned by calloc() is not null |
| #include <stdlib.h>  #include <string.h>    enum { SIG\_DESC\_SIZE = 32 };    typedef struct {  **char** sig\_desc[SIG\_DESC\_SIZE];  } signal\_info;    void func(**size\_t** num\_of\_records, **size\_t** temp\_num,            const **char** \*tmp2, **size\_t** tmp2\_size\_bytes) {    signal\_info \*start = (signal\_info \*)**calloc**(num\_of\_records,                                             sizeof(signal\_info));    if (start == NULL) {      /\* Handle allocation error \*/    } else if (tmp2 == NULL) {      /\* Handle error \*/    } else if (temp\_num > num\_of\_records) {      /\* Handle error \*/    } else if (tmp2\_size\_bytes < SIG\_DESC\_SIZE) {      /\* Handle error \*/    }      signal\_info \*point = start + temp\_num - 1;  **memcpy**(point->sig\_desc, tmp2, SIG\_DESC\_SIZE);    point->sig\_desc[SIG\_DESC\_SIZE - 1] = '\0';    /\* ... \*/  **free**(start);  } |

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

|  |
| --- |
| **Principles(s):** Practice Defense in Depth because you acknowledge the error and or try to prevent it so that the whole application do not crash. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Error-information-unused  Error-information-unused-computed | Partially checked |
| Axivion Bauhaus Suite | 6.9.0 | CertC-ERR33 | [Insert text.] |
| CodeSonar | 6.0p0 | LANG.FUNCS.IRV | Ignored return value |
| Compass/ROSE | [Insert text.] | [Insert text.] | Can detect violations of this recommendation when checking for violations of EXP-12-C. Do not ignore values returned by functions and EXP34-C. Don not dereference null pointers. |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
|  | [STD-008-CPP] |  |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| [Noncompliant code block; code should be indented using 12-point Courier New font.] |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-009-CPP] | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| [Noncompliant code block; code should be indented using 12-point Courier New font.] |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Student Choice] | [STD-010-CPP] | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| [Noncompliant description] |
| [Noncompliant code block; code should be indented using 12-point Courier New font.] |

| **Compliant Code** |
| --- |
| [Compliant description] |
| [Compliant code block; code should be indented using 12-point Courier New font.] |

**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** |
| --- | --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## 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.

[Insert your written explanations here.]

## 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-002-CPP | High | Probable | High | P6 | 2 |
| STD-003-CPP | High | Likely | Medium | P18 | 1 |
| STD-004-CPP | High | Likely | Medium | P18 | 1 |
| STD-005-CPP | Medium | Probable | Medium | P8 | L2 |
| STD-060-CPP | Medium | Unlikely | Medium | P4 | L3 |
| STD-070-CPP | High | Likely | Medium | P18 | L1 |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

## Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Data that is stored in a place or like a cloud. Validate Input Data it the proper testing of any input data supplied by a user application. So anything things that is inputted it goes through the proper checks so that things like memory leakage won’t happen. |
| Encryption at flight | It is data being moved from one place to another. In other words a data can be moved from one file to another. Sanitize Data sent to Other Systems sanitize all data and pass it through all subsystems before it is invoking the subsystem. |
| Encryption in use | When data is going from point A to point B. Also, in use could be handled in protected memory or the data can be transformed for use. Validate Input Data or data validation prevents improperly formed data from entering an information system. So it should occur when data is received from an external party that is not trusted if not checked before it was inputted it can compromised systems. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Is the process of recognizing a user’s identity. Default Deny is a decision based on permissions. This basically means that unless you give a device/user permission to the database it will not all access. |
| Authorization | Is a security mechanism to determine access levels or user/client privileges related to system resources including files, services, computer programs, data, and application features. Adhere to the Principle of least Privilege which is just giving access to the least number of users or giving least amount access to files so that it will reduce the chance of hackers. |
| Accounting | Incorporates conventional accounting standards with programming and data frameworks to make a brought together area for putting away a substance’s money related information. |

**\***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 |