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



Green Pace Secure Development Policy

Contents

[Overview](#bookmark) 2

[Purpose](#bookmark1) 2

[Scope](#bookmark2) 2

[Module Three Milestone](#bookmark3) 2

[Ten Core Security Principles](#bookmark4) 2

[C/C++ Ten Coding Standards](#bookmark5) 3

[Coding Standard 1](#bookmark6) 4

[Coding Standard 2](#bookmark7) 5

[Coding Standard 3](#bookmark8) 6

[Coding Standard 4](#bookmark9) 7

[Coding Standard 5](#bookmark10) 8

[Coding Standard 6](#bookmark11) 9

[Coding Standard 7](#bookmark12) 10

[Coding Standard 8](#bookmark13) 11

[Coding Standard 9](#bookmark14) 13

[Coding Standard 10](#bookmark15) 14

[Defense-in-Depth Illustration](#bookmark16) 15

[Project One](#bookmark17) 15

[1. Revise the C/C++ Standards](#bookmark18) 15

[2. Risk Assessment](#bookmark19) 15

[3. Automated Detection](#bookmark20) 15

[4. Automation](#bookmark21) 15

[5. Summary of Risk Assessments](#bookmark22) 16

[6. Create Policies for Encryption and Triple A](#bookmark23) 16

[7. Map the Principles](#bookmark24) 17

[Audit Controls and Management](#bookmark25) 18

[Enforcement](#bookmark26) 18

[Exceptions Process](#bookmark27) 18

[Distribution](#bookmark28) 19

[Policy Change Control](#bookmark29) 19

[Policy Version History](#bookmark30) 19

[Appendix A Lookups](#bookmark31) 19

[Approved C/C++ Language Acronyms](#bookmark32) 19

# 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 | We here at Green Pace rely upon a centralized validation model utilizing shared libraries and filtering code. We assume that all input is potentially malicious and screen all users input for correct format, length, range, and correct type, reject bad types and sanitize input. |
| 1. Heed Compiler Warnings | At Green Pace we use vast resources to procure great compiler software. Each programmer sets their compilers to the highest warning level available. We utilize static and dynamic analysis and testing techniques to eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | At Green Pace we design and create software architecture to implement and enforce security policies. For example: Our system requires different privileges at different times, so we have divided the system into distinct intercommunicating subsystems, each with and appropriate privilege |
| 1. Keep It Simple | At Green Peace, as a general philosophy we believe in the KISS (Keep It Simple Stupid) method. We regularly implement, and audit production software and architecture to keep the entire system running as simply and efficiently as possible. This ensures that our simplest safety precautions are also the safest. |
| 1. Default Deny | At Green Pace we base our access decisions on permission rather than exclusion. Essentially, everyone is excluded and we decide the parameters in which people are included in our system. |
| 1. Adhere to the Principle of Least Privilege | At Green Pace we try to ensure that everyone is able to get their job done, but with the least amount of permissions possible. Each elevated permission should only be permitted for the least amount of time required. This lessens the ability for any user to execute arbitrary code at any time. This also has the added benefit of knowing whom had which permissions at a given time, thus making it easier to identify who most likely was responsible for the breach. |
| 1. Sanitize Data Sent to Other Systems | Code will be regularly sanitized to ensure that all unused or superfluous code will be scraped before any data is passed to complex subsystems. This will help ward of attacks as each of those functions leave another area open for attack. |
| 1. Practice Defense in Depth | At Green Pace there are multiple layers of security starting with the personelle and specific training given to each employee on how to safeguard our data. In addition we have implemented hardware and software solutions to prevent physical breaches of each of our devices as well as compartmentalizing our data so that no sensitive data is housed in one specific location. |
| 1. Use Effective Quality Assurance Techniques | At Green Pace our code, department, and security solutions are regularly audited to ensure that we maintain nothing but the highest security standards. We regularly seek testers outside of our organization to see if they can penetrate or bypass our systems. |
| 1. Adopt a Secure Coding Standard | At Green Pace, it is our sincerest wish to provide each of our employees with a comprehensive guide in their approach to coding. It is with that intention that we are creating this document, to approach coding with sound principles and uniform precision, thus limiting our exposure to the greatest degree possible. |

## 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] | [incorrect.-type conversion] |

| **Noncompliant Code** |
| --- |
| When converting integers to pointers or pointers to integers and implemented poorly can lead to undesired consequences. This can happen when you:   1. Convert an integer to a pointer and the resulting pointer type is aligned incorrectly or fails to point to the referenced type 2. Convert a pointer to an integer when the resulting type cannot be correctly represented as an integer 3. Mapping between integers and pointers is inconsistent with the environment’s addressing structure   Below is an example of a vulnerability in which the code converts the pointer to an integer. The nine bits(number) are used for holding the flag value before the result is converted into a pointer. The code does not comply to the conversion rule because a 32 bit integer cannot represent a 64 bit pointer. |
| void func(unsigned int flag) {  char \*ptr;  /\* ... \*/  unsigned int number = (unsigned int)ptr;  number = (number & 0x7fffff) | (flag << 23);  ptr = (char \*)number;  } |

| **Compliant Code** |
| --- |
| Using “Struct” to store the pointer and flag value can help solve the issue. |
| struct ptrflag {  char \*pointer;  unsigned int flag : 9;  } ptrflag;    void func(unsigned int flag) {  char \*ptr;  /\* ... \*/  ptrflag.pointer = ptr;  ptrflag.flag = flag;  } |

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

|  |
| --- |
| **Principles(s):** Architecture and Design (This is a flaw during creation/implementation in which a data type was not allocated correctly which could result in a failure point within the system.) |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Unlikely | Medium | P9 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| LDRA tool suite | 9.7.1 | 296 S | Partially Implemented |
| Parasoft C/C++ Test | 2020.2 | CERT\_CPP-DCL53-a  CERT\_CPP-DCL53-b | Always declare functions at file scope  Identifier declared in a local function prototype scope shall not hid an identifier declared in a global or namespace |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-CPP] | [No Unsigned Integer Wrapping] |

| **Noncompliant Code** |
| --- |
| When implementing arithmetic operations make sure that you follow the rules of integer conversion, otherwise you may end up with an unsigned integer wrapping problem. Unsigned Integer Wrapping occurs when the underlying integer representation cannot be used in place of the resulting value. This can expose your system to all sorts of vulnerabilities if the values are used in:   1. Integer operands used in pointer arithmetic and array indexing 2. Postfix expression just before square brackets [] 3. Functional arguments to a memory allocation function 4. Assignment expressions used to declare a variable-length array 5. Code that is crtical to secutiry |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| To ensure no unsigned wrap occurs and minimize possible vulnerabilities a pre-condition test should be performed as shown here: |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum;  if (UINT\_MAX - ui\_a < ui\_b) {  //Handle error  } else {  usum = ui\_a + ui\_b;  }  /\* .......\*/  } |

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

|  |
| --- |
| **Principles(s):** [Overflow Prevention] (By not assigning each integer a negative or positive the system assumes the integer is positive, we should always specify] |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PRQA QA-C | 9.7 | 2910[C], 2911[D], 2912[A],  2913[S], 3383, 3384, 3385, 3386 | Partially implemented |
| PVS -Studio | 7.07 | V658, V1028 | N/A |
| TrustInSoft Analyzer | 1.38 | Unsigned overflow | Exhaustively verified |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-CPP] | [Invalid String Format] |

| **Noncompliant Code** |
| --- |
| Strings need to be formatted correctly. Not formatting strings correctly could lead to abnormal program termination, memory corruption or a whole bunch of unwanted behaviors.  Below is an example of an invalid string format. The code below has a mismatch between the error\_type argument and the S specifier. The error\_msg argument is incorrectly matched to the d specifier. This could eventually result in access violation because cout() interprets the error\_type argument as a pointer. |
| void func(void) {  const char \*error\_msg = "Resource not available to user.";  int error\_type = 3;  /\* ... \*/  cout("Error (type %s): %d\n", error\_type, error\_msg);  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| To fix this, we ensure that arguments to the cout() function are matched to the right conversion specifications as follows: |
| void func(void) {  const char \*error\_msg = "Resource not available to user.";  int error\_type = 3;  /\*code follows\*/    cout("Error (type %d): %s\n", error\_type, error\_msg);    /\*code follows\*/    } |

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

|  |
| --- |
| **Principles(s):** [**Architecture and Design, Authentication**] (The problem with accepting invalid user input is generally done within the implementation and design phases, and it is an authentication issue in which we should authenticate whether or not the users input is valid. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ParaSoft C/C++ test | [2020.1] | CERT\_CPP\_STR52a | Use valid pointers, iterators and references |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-CPP] | [Rationalize the standard.] |

| **Noncompliant Code** |
| --- |
| Prepared Statement |
| class Login {    public Connection getConnection() throws SQLException {      DriverManager.registerDriver(new              com.microsoft.sqlserver.jdbc.SQLServerDriver());      String dbConnection =        PropertyManager.getProperty("db.connection");      // Can hold some value like      // "jdbc:microsoft:sqlserver://<HOST>:1433,<UID>,<PWD>"      return DriverManager.getConnection(dbConnection);    }      String hashPassword(char[] password) {      // Create hash of password    }      public void doPrivilegedAction(      String username, char[] password    ) throws SQLException {      Connection connection = getConnection();      if (connection == null) {        // Handle error      }      try {        String pwd = hashPassword(password);        String sqlString = "select \* from db\_user where username=" +          username + " and password =" + pwd;        PreparedStatement stmt = connection.prepareStatement(sqlString);          ResultSet rs = stmt.executeQuery();        if (!rs.next()) {          throw new SecurityException("User name or password incorrect");        }          // Authenticated; proceed      } finally {        try {          connection.close();        } catch (SQLException x) {          // Forward to handler        }      }    }  } |

| **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):** Verification, Database Security (Relates to verification because we should verify that the input is expected, and database security as that is what would be compromised if SQL Injection were successful) |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| The Checker Framework | 2.1.3 | Tainting Checker | Trust and security errors |
| Coverity | 7.5 | SQLI , FB.SQL\_Prepared\_Statement\_Generated\_  FB.SQL\_NonConstant\_String\_passed\_to\_execute | Implemented |
| Findbugs | 1.0 | SQL\_Nonconstant\_string\_passed\_to\_execute | Implemented |
| Fortify | 1.0 | HTTP\_Response\_Splitting | Implemented |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-CPP] | [Buffer Copy Without Checking Size of Input] |

| **Noncompliant Code** |
| --- |
| Overflow can be prevented by controlling the number of characters read into the buffer. If the BUFF\_SIZE argument is larger than the size of the buffer, overflow will still occur. In this example, an overflow could have been. Prevented by limiting the size of the BUFF\_SIZE argument with an fgets function. |
| Printf(“Enter the master password: \n”);  Gets(userPass);  If(strcmp(userPass,MASTER\_PASSWORD)==0){  Printf(“PASSWORD VERIFIED\n”);  } |

| **Compliant Code** |
| --- |
| In this example there is no user overflow because the program checks to make sure no more than 9 characters are read. If the password equals the correct amount of characters, then the password is verified. This ensures that the correct amount of characters are used and no more. |
| Printf(“Enter the master password: \n”);  fgets(userPass, 9, stdin);  If(strcmp(userPass,MASTER\_PASSWORD, 9)==0){  Printf(“PASSWORD VERIFIED\n”);  } |

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

|  |
| --- |
| **Principles(s):** [Buffer overflow prevention, Authentication, |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PVS-Studio | 7.07 | V531, V635, V641, V781 | [Insert text.] |
| RuleCJecker | 20.10 | Malloc-size-insufficient | [Insert text.] |
| TrustInSoftAnalyzer | 1.38 | Mem-access | [Insert text.] |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-CPP] | Disable asserts after testing code |

| **Noncompliant Code** |
| --- |
| Assertions are used a lot to validate output, and while assertions make debugging efficient care should be taken not to include assertions in the release build of an application. Below is an example of an assertion being made and in the message displayed on the terminal, explains where the assertion failed. |
| #include <iostream>  #include <cassert>  using namespace std;  void display\_number(int\* myInt) {    assert (myInt!=NULL);    cout<<"myInt contains value" << " = "<<\*myInt<<endl;  }  int main ()  {    int myptr=5;    int \* second\_ptr = NULL;    int \* third\_ptr = NULL;    second\_ptr=&myptr;    display\_number (second\_ptr);    display\_number (third\_ptr);    return 0;  } |

| **Compliant Code** |
| --- |
| Using NDEBUG macro in a program disable all calls to assert. By implementing NDEBUG, the assert statements are no more active and continues its normal execution even when the second condition in the assert statement is false |
| #include <iostream>  // uncomment: assert() disabled  #define NDEBUG  #include <cassert>  using namespace std;  int main()  {      assert(2+2==3+1);      cout << "Expression valid...Execution continues.\n";      assert(2+2==1+1);      cout << "Assert disabled...execution continuous with invalid expression\n";  } |

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

|  |
| --- |
| **Principles(s):** [Architecture And Design, Error Handling] (Errors will occur and assertions are made to assess where those errors are, once the program is ready to be implemented errors should be less explicit) |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| PRQA QA-C++ | 4.4 | 3225, 3226, 3227, 3228, 3229 | N/A |
| Prasoft C/C++ test | 2020.2 | CERT\_C-PRE31-a  CERT\_C-PRE31-b  CERT\_C-PRE31-c  CERT\_C-PRE31-d | A full expression. Containing an increment(++) or decrement (--) operator should have no other potential side effects  Assertions should not contain assignments, increment or decrement operators  Assertions should not contain function calls nor function-like macro calls  Avoid side effects in arguments to unsafe macros |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-CPP] | Exceptions should not be ignored |

| **Noncompliant Code** |
| --- |
| When an exception occurs it is a bad idea to just ignore them. Instead, it is better to handle them properly or at least to log them. The code below catches the exception but doesn’t save it anywhere |
| void save() {  try {  saveDocument();  } catch (const std::exception& ex) {  }  } |

| **Compliant Code** |
| --- |
| The code below executes the same code as above. However, instead of ignoring the exception when it is thrown, catches and logs the exception to ex.what |
| void save() {  try {  saveDocument();  } catch (const std::exception& ex) {  log << "Exception while saving the document: " << ex.what();  }  } |

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

|  |
| --- |
| **Principles(s):** [Architecture and Design, Error handling] (During the design phase exceptions are created to log errors in the logic, these errors should be corrected to ensure functionality) |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ Test | 2020.2 | Cert\_CPP-ERR55-a | Where a function’s declaration includes an expception-specification, the function shall only be capable of throwing exceptions of the indicated type(s) |
| PRQA QA – C++ | 4.4 | 4035, 4036, 4632 | N/A |
| RuleChecker | 20.10 | Unhandled-throw-noexcept | Partially Checked |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [Account Validity] | [STD-008-CPP] | Account validity should be verified when authenticating users with PAM |

| **Noncompliant Code** |
| --- |
| When using Pluggable authentication module (PAM) it is strongly recommended to check the validity of the account(not locked, not expired), otherwise it leads to unauthorized access to resources. The code below is not checked with pam\_acct\_mgmt when authenticating a user with pam\_authenticate: |
| int valid(pam\_handle\_t \*pamh) {  if (pam\_authenticate(pamh, PAM\_DISALLOW\_NULL\_AUTHTOK) != PAM\_SUCCESS) { // Noncompliant - missing pam\_acct\_mgmt  return -1;  }  return 0;  } |

| **Compliant Code** |
| --- |
| [The code below does check with pam\_authenticate, and checks the account validity with pan\_acct\_mgmt: |
| int valid(pam\_handle\_t \*pamh) {  if (pam\_authenticate(pamh, PAM\_DISALLOW\_NULL\_AUTHTOK) != PAM\_SUCCESS) {  return -1;  }  if (pam\_acct\_mgmt(pamh, 0) != PAM\_SUCCESS) { // Compliant  return -1;  }  return 0;  } |

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

|  |
| --- |
| **Principles(s):** [Authentication, Architecture and Design] (Establishes best practices for ensuring secure sign on) |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | High | P12 | L2 |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [INT] | [STD-009-CPP] | [Don’t cast to an enumeration value this not in range] |

| **Noncompliant Code** |
| --- |
| In this example the code tries to check whether or not a given value is within the range of acceptable enumeration value, but it is doing so after casting to the enumeration type. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| Check the bounds, check that the value can be represented by the enumeration type before performing the conversion to guarantee the conversion does not result in an unspecified value |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

|  |
| --- |
| **Principles(s):** [Architecture and Design, Validation] Validates that the input of the data is within the parameters of the expected input |

**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 | CertC++ Into50 | N/A |
| Parasoft C/C++Test | 2020.2 | CERT\_CPP-Int50-a | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PRQA QA-C++ | 4.4 | 3013 | N/A |
| PVS-Studio | 7.07 | V1016 | N/A |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| [File] | [STD-010-CPP] | Close Files when they are no longer needed |

| **Noncompliant Code** |
| --- |
| Std::Fstream object file is constructed. The object files calls std::basic\_filebuf&ltlT&gt;::open(), which leaves the file open when std::terminate() is invoked. Thus, the file is not properly closed |
| # include <exception>  # include <fstream>  # include <string>  void f(**const** std::string &filename) {    std::fstream file(filename);  if (!file.is\_open()){  //Handle error  Return;  }  // …  Std::terminate();  } |

| **Compliant Code** |
| --- |
| In this code the std::fstream::close() is called before std::terminate() is called, ensuring that the file resources are properly closed: |
| # include <exception>  # include <fstream>  # include <string>  void f(**const** std::string &filename) {    std::fstream file(filename);  if (!file.is\_open()){  //Handle error  Return;  }  // …  File.close();  If (file.fail()) {  // Handle error  }  Std::terminate(); |

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

|  |
| --- |
| **Principles(s):** [Architecture and Design] improperly closed files could result in errors |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | ALLOC.LEAK | Leak |
| Klocwork | 2018 | RH.LEAK | N/A |
| Parasoft C/C++test | 2020.2 | CERT\_CPP\_fI051-a | Ensure resources are freed |
| Parasoft Insure++ | N/A | N/A | Runtime detection |

## 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.

[

As we continue our DevSecOps journey here at Green Pace, it is important to improve our methods and processes. In the Assess and Plan section we should include Threat Modeling Software to assess and make recommendations based off of the applications architecture, behavior and technology stack. By implementing this type of solution we improve our understanding of not only which threats pose the greatest danger to our applications, but also receive recommendations how to resolve those issues.

In addition, I would suggest that we implement Automated Dynamic Scans In The Nightly Builds(Continuous Integration Testing) in addition to scanning for vulnerabilities, we can more test the system more dynamically by building automated tests that run every evening. We could implement fuzzing tests that act like a person would act (i.e. trial and error) as if they had no intimate knowledge of the system in which they are trying to gain access to. Nightly automated testing ensures that we are able to scan for changes as they occur rather than scanning for vulnerabilities after every build or version is complete.

]

## 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 | P9 | L2 |
| [STD-002-CPP] | High | Likely | High | P9 | L2 |
| [STD-003-CPP] | High | Likely | Medium | P18 | L1 |
| [STD-004-CPP] | High | Probable | Medium | P12 | L1 |
| [STD-005-CPP] | High | Likely | High | P6 | L2 |
| [STD-006-CPP] | Medium | Probable | Medium | P12 | L1 |
| [STD-007-CPP] | Low | Likely | Low | P9 | L2 |
| [STD-008-CPP] | High | Medium | High | P12 | L1 |
| [STD-009-CPP] | Medium | Unlikely | Medium | P4 | L3 |
| [STD-0010-CPP] | Medium | Unlikely | Medium | P4 | L3 |

## 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 | Encryption at rest refers to data where it is stored. (i.e. computer, phone, database, etc…) |
| Encryption at flight | Encryption in use protects data as it is in transit from one location to another. |
| Encryption in use | As data is being created, the data itself is encrypted while being created. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authenticating a user is a step taken to ensure that only those that should have access to a system, does have access to the system. Examples of Authentication are two-step verification, email verification, and SMS verification. A newly onboarded employee will receive their credentials via their employee email which will be available upon day of hire.   * A New user will log in with their given credentials * The new user will then create their own unique password, 10 characters, 1 Uppercase, 1 lowercase, 1 special character, 1 number (no birthdays, phone numbers, addresses, etc.) * Each user will have to log in to both the username and password blocks each login * Users will have to reset their passwords every 3 months (this will cascade to all other logins as well) * If a user has forgotten their login credentials the user will be required to reset their password via email, and SMS authentication message. |
| Authorization | Authorization means that each user has specific permissions depending on their roles. Access should be limited to a specified period of time with specific permissions depending upon the users’ needs. For example, some developers may need access to IDE software, associated files for a specific project, database warehouse objects specific to that project, etc… once the project is completed, the developers’ permissions should then be revoked after the project has been completed.   * Initial Authorization begins with only read permissions on all files. * Once onboarded, each user will be given the permissions depending on their job title and department of assignment. * Should additional permissions be needed a form will be submitted by the employees' supervisor listing the Name of the supervisor, Name of the Project, Date of the request, Specified period of new authorizations, and a detailed list of which permissions need to be changed, and the reason for each change. |
| Accounting | Understanding who, or what entities at any given point in time entered or accessed any given system is important. This means logging changes, updates or new content created within the system or associated databases. This will ensure that any activity not allowed will not only be logged but flagged for immediate review.   * Each system accessed will require authentication and the system will check user permissions before allowing access. * Each users access will be logged in a daily\_users\_summary\_table. This will include length of time accessed, date, number of total commits, number of total downloads, and a concatenated list of the files accessed during each session. This will be one entry per day per user * Each system will also have a daily\_user\_detail table displaying which files were accessed, the time stamp, and the changes/updates or new files created in the system. |

**\***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:

1. Operating system logs
2. Firewall logs
3. Anti-malware logs
4. Authentication
5. Verification
6. Database Security
7. Overflow Prevention
8. Architecture And Design
9. Error Handling

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

# Audit Controls and Management

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

Evidence will include the following:

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

# Enforcement

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

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

# Exceptions Process

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

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

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

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

# Distribution

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

# Policy Change Control

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

# Policy Version History

| **Version** | **Date** | **Description** | **Edited By** | **Approved By** |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 2.0 | 06/07/2021 | [Insert text.] | Joshua Gierlich | [Insert text.] |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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