**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 | Be suspicious and validate input of untrusted data sources. This could be command line arguments, environmental variables and user-controlled files. |
| 1. Heed Compiler Warnings | Compile code using the highest warning level for the compiler. Also, use static and dynamic analysis tools to detect security flaws. |
| 1. Architect and Design for Security Policies | Design architecture that enforces security policies. |
| 1. Keep It Simple | The more complex as system the better chance of errors during implementation, configuration, and use. |
| 1. Default Deny | Default access should be denied, and to gain access conditions should be met to be permitted. |
| 1. Adhere to the Principle of Least Privilege | Every process should execute with the least number of privileges and every elevated privilege should be accessed for the least amount of time. |
| 1. Sanitize Data Sent to Other Systems | Sanitize all data sent to the subsystems such as command shells, databases and off the shelf products. |
| 1. Practice Defense in Depth | Manage risks in multiple layers, if one layer is not working properly another layer can stop an exploit. |
| 1. Use Effective Quality Assurance Techniques | Make use of many different quality assurance techniques such as: Fuzz testing, source code audits, penetration testing and external audits. |
| 1. Adopt a Secure Coding Standard | Secure coding standards should be created at the beginning of every project so the team has is less likely to write code that can become easily exploited. |

## 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-DATTYP-C | Make sure unsigned integer operations do not wrap. |

| **Noncompliant Code** |
| --- |
| An unsigned integer wrap during the addition of the unsigned operands ui\_a and ui\_b. This could result in insufficient memory and can lead to a vulnerability. |
| void func(unsigned int ui\_a, unsigned int ui\_b) {  unsigned int usum = ui\_a + ui\_b;  /\* ... \*/  } |

| **Compliant Code** |
| --- |
| Solution performs a precondition test of the operands of the addition to guarantee there is no possibility of an unsigned wrap. |
| #include <limits.h>    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):** Architect and design for security policies. Write code to that performs the precondition test to make sure there is no unsigned wrap. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | INTEGER\_OVERFLOW | Implemented |
| Astree | 20.10 | Integer-overflow | Fully checked |
| LDRA tool suite | 9.7.1 | 493 S, 494 S | Partially Implemented |
| TrustInSoftAnalyzer | 1.38 | Unsigned overflow | Exhaustively verified |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-001-DAT-C | Do not compare padding data. |

| **Noncompliant Code** |
| --- |
| Memcmp () is used to compare the contents of two structures, including padding bytes: |
| #include <string.h>    struct s {  char c;  int i;  char buffer[13];  };    void compare(const struct s \*left, const struct s \*right) {  if ((left && right) &&  (0 == memcmp(left, right, sizeof(struct s)))) {  /\* ... \*/  }  } |

| **Compliant Code** |
| --- |
| All the fields are compared manually to avoid comparing any padding bytes: |
| #include <string.h>    struct s {  char c;  int i;  char buffer[13];  };  void compare(const struct s \*left, const struct s \*right) {  if ((left && right) &&  (left->c == right->c) &&  (left->i == right->i) &&  (0 == memcmp(left->buffer, right->buffer, 13))) {  /\* ... \*/  }  } |

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

|  |
| --- |
| **Principles(s):** Practice Defense in Depth. It is better to manually compare all the fields to avoid any padding bytes. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Memcpy-with-padding | Partially checked |
| Axivion Bauhaus Suite | 2021.2 | CertC-EXP42 | Implemented |
| LDRA tool suite | 9.7.1 | 618 S | Partially implemented |
| Polyspace Bug Finder | R2021a | CERT C: Rule EXP42-C | Checks for memory comparison of padding data. |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-001-STR - C | Do not attempt to modify string literals |

| **Noncompliant Code** |
| --- |
| The char pointer to str is initialized to the address of a string literal, which attempting to modify the string literal is undefined behavior. |
| **char** \*str  = "string literal";  str[0] = 'S'; |

| **Compliant Code** |
| --- |
| This code creates a copy of the string literal in the space allocated to the character array str. The string stored can be modified. |
| char str[] = "string literal";  str[0] = 'S'; |

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

|  |
| --- |
| **Principles(s):** Keep it simple. Use an array instead of a pointer to allocate a string and be able to modify it later. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Likely | Low | P9 | Level 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| LDRA tool suite | 9.7.1 | 157 S | Partially Implemented |
| RuleChecker | 20.10. | String-literal-modification | Partially checked |
| Polyspace Bug Finder | R2021a | Cert C: RuleSTR30-C | Checks for writing to const qualified object. (Rule fully covered) |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-001-SQLINJ - J | Prevent SQL INJECTION |

| **Noncompliant Code** |
| --- |
| SQL injection can happen because of unsanitized input argument username into SQL command, allowing an attacker to inject validuser or 1=1. |
| import java.sql.Connection;  import java.sql.DriverManager;  import java.sql.ResultSet;  import java.sql.SQLException;  import java.sql.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 + "'";  Statement stmt = connection.createStatement();  ResultSet rs = stmt.executeQuery(sqlString);    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** |
| --- |
| This code validates the length of username argument, preventing an attacker from submitting an arbitrarily long username. |
| public void doPrivilegedAction(  String username, char[] password  ) throws SQLException {  Connection connection = getConnection();  if (connection == null) {  // Handle error  }  try {  String pwd = hashPassword(password);    // Validate username length  if (username.length() > 8) {  // Handle error  }    String sqlString =  "select \* from db\_user where username=? and password=?";  PreparedStatement stmt = connection.prepareStatement(sqlString);  stmt.setString(1, username);  stmt.setString(2, pwd);  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  }  }  } |

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

|  |
| --- |
| **Principles(s):** Default Deny, Validate input data. SQL injections can happen from writing improper code for logins. Make sure that the code does not allow an injection. |

**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 |
| CodeSonar | 6.1p0 | Java.IO.INJ.SQL | SQL Injection |
| SonarQube | 6.7 | S2077, S3649 | Executing SQL queries is security sensitive |
| Parasoft Jtest | 2021.1 | CERTIDSOO>TDSQL | Protect against SQL injection |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-001-MEM-C | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| P is freed before p->next is executed, p-> next reads memory that has already been freed. |
| #include <stdlib.h>    struct node {  **int** value;    struct node \*next;  };    void free\_list(struct node \*head) {    for (struct node \*p = head; p != NULL; p = p->next) {  **free**(p);    }  } |

| **Compliant Code** |
| --- |
| Storing a reference to p->next in q before free p is correct. |
| #include <stdlib.h>    struct node {  int value;  struct node \*next;  };    void free\_list(struct node \*head) {  struct node \*q;  for (struct node \*p = head; p != NULL; p = q) {  q = p->next;  free(p);  }  } |

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

|  |
| --- |
| **Principles(s):** Use effective quality assurance techniques. Many automated tools will detect freed memory. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | dangling\_pointer\_use | Supported |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM30 | Detects memory accesses after its deallocation and double memory deallocations |
| Coverity | 2017.07 | USE\_AFTER\_FREE | Can detect specific instances where memory is deallocated more than once or read/written to the target of a freed pointer. |
| LDRA Tool Suite | 9.7.1 | 51 D, 484 S, 112 D | Partially implemented |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-001-ASSERT-C | Run diagnostic tests using assertions |

| **Noncompliant Code** |
| --- |
| Using assert() macro to verify that memory allocation succeeded could lead to the termination of process and open the possibility of a denial-of-service attack. |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;    len = strlen(c\_str);  dup = (char \*)malloc(len + 1);  assert(NULL != dup);    memcpy(dup, c\_str, len + 1);  return dup;  } |

| **Compliant Code** |
| --- |
| This code detects and handles possible memory exhaustion. |
| char \*dupstring(const char \*c\_str) {  size\_t len;  char \*dup;    len = strlen(c\_str);  dup = (char\*)malloc(len + 1);  /\* Detect and handle memory allocation error \*/  if (NULL == dup) {  return NULL;  }    memcpy(dup, c\_str, len + 1);  return dup;  } |

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

|  |
| --- |
| **Principles(s): Use quality assurance techniques, Heed compiler warnings. It is important to test your code thoroughly before releasing it.** |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | LANG.FUNCS.ASSERTS | Not enough assertions |
| Coverity | 2017.07 | ASSERT\_SIDE\_EFFECT | Can detect the specific instance where assertion contains an operation/function call that may have a side effect |
| Parasoft C/C++test | 2021.1 | CERT\_C\_MSC11-a | Assert liberally to document internal assumptions and invariants |
| N/A | N/A | N/A | N/A |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-001-EXC-CPP | Honor exception specifications |

| **Noncompliant Code** |
| --- |
| A function is declared nonthrowing, but it is possible for std::vector::resize() to throw an exception when the requested memory cannot be allocated |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) noexcept(true) {  v.resize(s); // May throw  } |

| **Compliant Code** |
| --- |
| The function’s noexcept-specification is removed, now the function allows all exceptions. |
| #include <cstddef>  #include <vector>    void f(std::vector<int> &v, size\_t s) {  v.resize(s); // May throw, but that is okay  } |

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

|  |
| --- |
| **Principles(s):** Architect and Design for Security Policies, Use Effective Quality Assurance Techniques, Heed compiler warnings. Exceptions help error handling code. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Unhandled-throw-noexcept | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CERTC++-ERR55 | No description |
| LDRA Tool Suite | 9.7.1 | 56D | Partially Implemented |
| RuleChecker | 20.10 | unhandled\_throw-noexcept | Partially checked |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Arrays** | STD-001-ARR-C | Explicitly specify array bounds, even if implicitly defined by an initializer |

| **Noncompliant Code** |
| --- |
| This code initializes an array of integers using an initialization with too many elements for the array. |
| int a[3] = {1, 2, 3, 4}; |

| **Compliant Code** |
| --- |
| This code explicitly specifies the array boundary. |
| int a[4] = {1, 2, 3, 4}; |

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

|  |
| --- |
| **Principles(s):** Keep it simple, Heed compiler Warnings. It is good practice to specify array boundaries. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Unlikely | Low | P6 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | array-size-global | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-ARR02 | Fully implemented |
| CodeSonar | 6.1p0 | LANG.STRUCT.DECL.FAM | Declaration of flexible array member |
| ÉCLAIR | 1.2 | CC2.ARR02 | Fully implemented |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Declaration and Initialization** | STD-001-DCL-C | Do not declare more than one variable per declaration. |

| **Noncompliant Code** |
| --- |
| Declares more than one variable at once. |
| char \*src = 0, c = 0; |

| **Compliant Code** |
| --- |
| Each variable is declared on a separate line. |
| char \*src; /\* Source string \*/  char c; /\* Character being tested \*/ |

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

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

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.1p0 | LANG.STRUCT.DECL.ML | Multiple Declarations on Line |
| ÉCLAIR | 1.2 | CC2.DCL04 | Fully implemented |
| LDRA Tool Suite | 9.7.1 | 579 S | Fully Implemented |
| PC-lint Plus | 1.4 | 9146 | Partially supported: exceptions not supported |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String** | STD-002-STR-C | Do not concatenate different types of string literals. |

| **Noncompliant Code** |
| --- |
| Code concatenates wide and narrow string literals. |
| wchar\_t \*msg = L"This message is very long, so I want to divide it "  "into two parts."; |

| **Compliant Code** |
| --- |
| Each element in the concatenation must be a wide string literal. |
| wchar\_t \*msg = L"This message is very long, so I want to divide it "  L"into two parts."; |

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

|  |
| --- |
| **Principles(s):** Architect and design for secure policies, keep it simple. To keep code error proof use the right type of string literals while concatenating. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | encoding-mismatch | Fully checked |
| ÉCLAIR | 1.2 | CC2.STR10 | Fully implemented |
| LDRA Tool Suite | 2021.2 | C0874 | Fully Implemented |
| PC-lint Plus | 1.4 | 707 | Fully Supported |

## 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.

Security is a major priority and automation tools can help code be tested many times a day. Automated testing should be done during the entire development lifecycle as soon as there is code to test. Automation in the DevSecOps should be tested during the design phase. This is also where in the rubric it states IDE security plug-ins pointing towards create. The automated testing should continue until for the rest of the DevSecOps Toolchain.

## 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-001-DATTYP-C | High | Likely | High | High | 2 |
| STD-001-DAT-C | Medium | Probable | Medium | Medium | 2 |
| STD-001-STR - C | Low | Likely | Low | Medium | 2 |
| STD-001-SQLINJ - J | High | Probable | High | High | 1 |
| STD-001-MEM-C | High | Likely | Medium | High | 1 |
| STD-001-ASSERT-C | Low | Unlikely | High | Low | 3 |
| STD-001-EXC-CPP | Low | Likely | Low | Medium | 2 |
| STD-001-ARR-C | Medium | Unlikely | Low | 6 | 2 |
| STD-001-DCL-C | Low | Unlikely | Low | 3 | 3 |
| STD-002-STR-C | Low | Probable | Medium | 4 | 3 |

## Create Policies for Encryption and Triple A

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

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

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

| 1. **Encryption** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Encryption in rest | Encryption at rest is designed to prevent an attacker from getting data while on disk. There is a symmetric encryption key to encrypt data as it’s written in storage. The same key is used to decrypt the data. Keys must be stored in a secure location with access control. This policy helps protect data if a hard-drive or other physical storages are stolen. |
| Encryption at flight | Encryption in flight is encryption while transmitting data. With the extensive use of the cloud this is very important. This is important when sending data to storage or transmitting data to a customer. |
| Encryption in use | In-use Encryption ensures sensitive data is secure regardless of the lifecycle or location. It is Role-based with access controls to control which users can access specific data. You can also use it for all data stores from a single platform and it detects anomalies in real time. This policy can be applied at the beginning of the development lifecycle and used throughout. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the way a network or application identifies a user. Usually there is a username and a password before access is granted. Usually, the username and password are stored in a database and once logged in the user can use the software or network. This policy applies because it is important to authenticate regardless of if you are an employee or customer to make sure there is data integrity. |
| Authorization | Users have different levels of authorization for what resources they can access. An administrator could add users and change passwords. A regular user might only be able to read certain documents. Some files might not be able to be accessed if the user is not authorized. This is an important policy because users only need enough access to get the job done. |
| Accounting | Accounting is measuring resources of what users use when accessing networks and applications. Administrators can see the actions that user performs. This is important because it shows who has been attempting to access resources and what resources they were trying to access. |

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

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

## Map the Principles

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

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

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

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

# Audit Controls and Management

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

Evidence will include the following:

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

# Enforcement

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

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

# Exceptions Process

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

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

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

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

# Distribution

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

# Policy Change Control

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

# Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 08/10/2021 | Standards | Stephen Blackburn |  |
| 1.2 | 08/12/2021 | Finished Policy | Stephen Blackburn |  |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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

References

*5 facts you probably didn’t know about in-flight data encryption*. (2020, May 22). Ciena - A networking systems, services, and software company - Ciena. <https://www.ciena.com/insights/articles/5-Facts-You-Probably-Didnt-Know-About-In-Flight-Data-Encryption.html>

Msmbaldwin. (n.d.). *Azure data encryption-at-Rest - Azure security*. Developer tools, technical documentation and coding examples | Microsoft Docs. <https://docs.microsoft.com/en-us/azure/security/fundamentals/encryption-atrest>

Mylonas, L. (2018, November 27). *What is AAA security? An introduction to authentication, authorisation and accounting*. Codebots. <https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting>

*Protect sensitive data | Data in use encryption | Tech brief*. (2021, June 4). Sotero. <https://www.soterosoft.com/blog/data-in-use-encryption-data-in-motion-encryption/>

*Rules explorer*. (n.d.). Rules explorer. <https://rules.sonarsource.com/cpp>

*STR53-CPP. Range check element access - SEI CERT C++ coding standard - Confluence*. (n.d.). <https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access>

Vijayan, J. (2017, December 13). *6 DevSecOps best practices: Automate early and often*. TechBeacon. STR53-CPP. Range check element access - SEI CERT C++ coding standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access