**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 | Validating Input data is first and foremost important step. Input from all the sources must be validate before using. This will help to minimize the overflow and other exception. |
| 1. Heed Compiler Warnings | [Head compiler warnings should not be ignored and can be solved before debugging. The compiler warnings can help the program to find the fault and minimize the risk of error. |
| 1. Architect and Design for Security Policies | This is used to describe and define how the system works. The Security policies both explicit and implicit should e document. Without these policies, the system may run to errors and flaws. |
| 1. Keep It Simple | The simpler it is, the better it is. The program should be simple so every programmer after you can understand it. |
| 1. Default Deny | Denying all the unnecessary traffic in the firewall and leaving it to user either block or accept. |
| 1. Adhere to the Principle of Least Privilege | Only giving specific job-related permission and blocking all the access content, app or programs to run. |
| 1. Sanitize Data Sent to Other Systems | Sanitizing means removing the unwanted data from the system to send tit to other system to protect user security. |
| 1. Practice Defense in Depth | Buffer overflows can occur and cause many companies at risk. But, using proper investigation and using good practice of Defense in depth we can try to protect our company data. |
| 1. Use Effective Quality Assurance Techniques | Keeping in mind the use of effective quality assurance, and user’s needs. The techniques might include QA team, testing, developing while testing, code review, user needs etc. |
| 1. Adopt a Secure Coding Standard | The standards must be used to protecting the software from attacks. |

### 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** | [DCL58-CPP] | Do not modify the standard namespace |

| **Noncompliant Code** |
| --- |
| In this example, the namespace has been used twice, while the integer x has been added to namespace. |
| #include <iostream>  Using namespace std;  Int main(){  Int x;  return 0;  }  Namespace std{  Int x;  } |

| **Compliant Code** |
| --- |
| To prevent from getting error or exception, we will change the code. |
| #include <iostream>  //Using namespace std;  Int main(){  Int x;  return 0;  }  Namespace nonstd{  Int x;  } |

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

| **Principles(s):** Do not modify the standard namespace |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Parasoft C/C++ test | 2021.1 | CERT\_CPP-DCL58a | Do not modify the standard namespaces 'std' and 'posix' |
| Polyspace Bug Finder | R2021b | CERT C++: DCL58-CPP | Checks for modification of standard namespaces (rule fully covered) |
| PVS-Studio | 7.15 | V1061 | Checker |
| SonaeQube c/c++ plugin | 4.10 | S3470 | New release |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [EXP40-C] | Do not modify constant objects |

| **Noncompliant Code** |
| --- |
| The constant integer was modified from 10 tp 0. |
| #include <iostream>  Using namespace std;  Const int x = 10;  Void func(){  X= 0;  } |

| **Compliant Code** |
| --- |
| This can be simply resolved by removing the const sign. |
| #include <iostream>  Using namespace std;  int x = 10;  Void func(){  X= 0;  } |

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

| **Principles(s):** Do not modify constant objects |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Medium | P2 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | assignment-to-non-modifiable-lvalue  pointer-qualifier-cast-const  pointer-qualifier-cast-const-implicit  write-to-constant-memory | Fully checked |
| Coverity | 2017.07 | PW  MISRA C 2004  Rule 11.5 | Implemented |
| LDRA tool suite | 9.7.1 | 582 S | Fully implemented |
| PRQA QA-C | 9.7 | 0563 | Partially implemented |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STR30-C | Do not attempt to modify string literals. |

| **Noncompliant Code** |
| --- |
| In the following code, the unassigned string modified to signed string. |
| Unassign str = “Hello!”;  Str = ‘s’;  While str = “Hello!”;  Return true; |

| **Compliant Code** |
| --- |
| This can be solved by using the following code: |
| Unassign str = “Hello!”;  Str = ‘s’;  While unassign str = “Hello!”;  Return true; |

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

| **Principles(s):** Do not attempt to modify string literals. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | string-literal-modfication  write-to-string-literal | Fully checked |
| Aixivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| Compass |  |  | Can detect simple violation |
| Parasoft C/C++ tool | 2021.1 | CERT\_C-STR30-a  CERT\_C-STR30-b | A string literal shall not be modified |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | IDS00-J | Prevent SQL injection |

| **Noncompliant Code** |
| --- |
| This will be an untrusted data formatting |
| 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      } |

| **Compliant Code** |
| --- |
| We solved this problem by throwing SQL injection |
| 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  } |

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

| **Principles(s):** Prevent SQL 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 | Trusted |
| CodeSonar | 6.1p0 | JAVA.IO.INJ.SQL | SQL injection |
| SonarQube | 6.7 | S2077  S3649 | Executing SQL |
| Fortify | 1.0 | HTTP\_Response\_Splitting | Implemented |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [MEM52-CPP] | Detect and handle memory allocation errors |

| **Noncompliant Code** |
| --- |
| Performing two memory allocations with same expression. |
| struct A { /\* ... \*/ };  struct B { /\* ... \*/ };    void g(A \*, B \*);  void f() {  g(new A, new B);  } |

| **Compliant Code** |
| --- |
| This can be solved by using unique pointer. |
| #include <memory>    struct A { /\* ... \*/ };  struct B { /\* ... \*/ };    void g(std::unique\_ptr<A> a, std::unique\_ptr<B> b);  void f() {  g(std::make\_unique<A>(), std::make\_unique<B>());  } |

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

| **Principles(s):** Detect and handle memory allocation errors |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | CHECKED\_RETURN | Finds inconsistencies in how function call return values are handled |
| LDRA tool suite | 9.7.1 | 45 D | Partially implemented |
| Polyspace Bug Finder | R2021b | CERT C++: MEM52-CPP | Checks for unprotected dynamic memory allocation (rule partially covered) |
| Parasoft Insure++ |  |  | Runtime detection |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | MET01-J | Never use assertions to validate method arguments |

| **Noncompliant Code** |
| --- |
| Sometimes, not providing the exact argument can result in assertions. |
| [public static int getAbsAdd(int x, int y) {  return Math.abs(x) + Math.abs(y);  }  getAbsAdd(Integer.MIN\_VALUE, 1); |

| **Compliant Code** |
| --- |
| This is taken from the provided source. |
| [public static int getAbsAdd(int x, int y) {  if (x == Integer.MIN\_VALUE || y == Integer.MIN\_VALUE) {  throw new IllegalArgumentException();  }  int absX = Math.abs(x);  int absY = Math.abs(y);  if (absX > Integer.MAX\_VALUE - absY) {  throw new IllegalArgumentException();  }  return absX + absY;  } |

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

| **Principles(s):** Never use assertions to validate method arguments |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| MITRE CWE | SWE-617 |  | Reachable assertion |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR59-CPP | Do not throw an exception across execution boundaries |

| **Noncompliant Code** |
| --- |
| Sometimes, the code throws an exception |
| #include<iosteream>  Using namespace std;  Int main(){  If (…){  Return true;  } |

| **Compliant Code** |
| --- |
| This can be prevented by using appropriate libraries. |
| #include<iosteream>  #Include <file>  Using namespace std;  Int main(){  If (…){  Return true;  } |

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

| **Principles(s):** Do not throw an exception across execution boundaries |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Helix QAC | 2021.2 | C + +3809,  C++3810 |  |
| Parasoft c/c++ teset | 2021.1 | CERT\_CPP-ERR59-a | Do not throw exception |

#### 

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Format settings | FIO47-C | Use valid format strings |

| **Noncompliant Code** |
| --- |
| Mismatching the argument type and conversion specifier |
| Int I = 12;  If(I = 1.2){  Return true;  } |

| **Compliant Code** |
| --- |
| This is resolved by changing int to float. |
| float I = 12;  If(I = 1.2){  Return true;  } |

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

| **Principles(s):** Use valid format strings |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO7 | Fully implemented |
| Coverity | 2017.07 | PW | Reports when the number of arguments differs from the number of required arguments according to the format string |
| CodeSonar | 6.1p0 | IO.INJ.FMT  MISC.FMT  MISC.FMTTYPE | Format string injection  Format string  Format string type error |
| Helix QAC | [2021.2 Insert text.] | C0161, C0162, C0163, C0164, C0165, C0166, C0167, C0168, C0169, C0170, C0171, C0172, C0173, C0174, C0175, C0176, C0177, C0178, C0179, C0180, C0184, C0185, C0190, C0191, C0192, C0193, C0194, C0195, C0196, C0197, C0198, C0199, C0200, C0201, C0202, C0204, C0206, C0209  C++3150, C++3151, C++3152, C++3153, C++3154, C++3155, C++3156, C++3157, C++3158, C++3159 |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Arrays | ARR32-C | Ensure size arguments for variable length arrays are in a valid range |

| **Noncompliant Code** |
| --- |
| Declaring the array size and then reusing it in different format. |
| { /\* Block scope \*/  char vla[size];  } |

| **Compliant Code** |
| --- |
| Error can be resolved using proper range. |
| { /\* Block scope \*/  char vla[size];  } |

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

| **Principles(s):** Ensure size arguments for variable length arrays are in a valid range |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| TrustInSoft Analyzer | 1.38 | alloca\_bounds | Exhaustively verified. |
| Polyspace Bug Finder | R2021a | CERT C: Rule ARR32-C | Checks for:  Memory allocation with tainted size  Tainted size of variable length array  Rule fully covered. |
| Parasoft C/C++test | 2021.1 | CERT\_C-ARR32-a | Ensure the size of the variable length array is in valid range |
| PC-lint Plus | 1.4 | 9035 | Assistance provided |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declaration | DCL41-C | Do not declare variables inside a switch statement before the first case label |

| **Noncompliant Code** |
| --- |
| The following code declares the expr in switch statement (source) |
| #include <stdio.h>    extern void f(int i);    void func(int expr) {  switch (expr) {  int i = 4;  f(i);  case 0:  i = 17;  /\* Falls through into default code \*/  default:  printf("%d\n", i);  }  } |

| **Compliant Code** |
| --- |
| Source |
| #include <stdio.h>    extern void f(int i);    int func(int expr) {  /\*  \* Move the code outside the switch block; now the statements  \* will get executed.  \*/  int i = 4;  f(i);    switch (expr) {  case 0:  i = 17;  /\* Falls through into default code \*/  default:  printf("%d\n", i);  }  return 0;  } |

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

| **Principles(s):** Do not declare variables inside a switch statement before the first case label |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 20.10 | Switch-skipped-code | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-DCL41 | Fully implemented |
| Clang | 3.9 |  |  |
| LDRA tool | 9.7.1 | 385 S | Fully implemented |

### 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.

In Green Pace, every aspect of Devops is considered under observation. This is particularly in reproduction cycle where everything renovate.

### 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 |
| DCL58-CPP | High | Unlikely | Medium | P6 | L2 |
| EXP40-C | Low | Unlikely | Medium | P2 | L3 |
| STR30-C | Low | Likely | Low | P9 | L2 |
| IDS00-J | High | Probable | Medium | P12 | L1 |
| MEM52-CPP | High | Likely | Medium | P18 | L1 |
| MET01-J | Medium | Probable | Medium | P8 | L2 |
| ERR59-CPP | High | Probable | Medium | P12 | L1 |
| FIO47-C | High | Unlikely | Medium | P6 | L2 |
| ARR32-C | Medium | Likely | M | P6 | L2 |
| DCL41-C | Medium | Unlikely | M | 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 is used to see if a particularly file is working. A switch statement causes control to jump to, into, or past the statement that is the switch body, depending on the value of a controlling expression, and on the presence of a default label and the values of any case labels on or in the switch body. |
| Encryption at flight | This means encryption is ready to be in launching mode |
| Encryption in use | After rest and flight the encryption finally took place. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | The details and documents related to production. |
| Authorization | Having access to all the sensitive things |
| Accounting | Making certain people in charge. |

**\***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 |