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



Green Pace Secure Development Policy

Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

[C/C++ Ten Coding Standards 3](#_Toc52464058)

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

[Defense-in-Depth Illustration 15](#_Toc52464069)

[Project One 15](#_Toc52464070)

[1. Revise the C/C++ Standards 15](#_Toc52464071)

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

[5. Summary of Risk Assessments 16](#_Toc52464075)

[6. Create Policies for Encryption and Triple A 16](#_Toc52464076)

[7. Map the Principles 17](#_Toc52464077)

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

[Approved C/C++ Language Acronyms 19](#_Toc52464085)

# Overview

Software development at Green Pace requires consistent implementation of secure principles to all developed applications. Consistent approaches and methodologies must be maintained through all policies that are uniformly defined, implemented, governed, and maintained over time.

# Purpose

This policy defines the core security principles; C/C++ coding standards; authorization, authentication, and auditing standards; and data encryption standards. This article explains the differences between policy, standards, principles, and practices (guidelines and procedure): [Understanding the Hierarchy of Principles, Policies, Standards, Procedures, and Guidelines](https://www.linkedin.com/pulse/understanding-hierarchy-principles-policies-standards-wally-beddoe/).

# Scope

This document applies to all staff that create, deploy, or support custom software at Green Pace.

# Module Three Milestone

## Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | Input validation is the first line of defense in secure coding. Input validation is checking the code that is inserted by users to check and make sure it is what we expect it to be. If it’s not what we want/expect then we return an error to the user with details on what we want/need. |
| 1. Heed Compiler Warnings | Compiler warnings are another one of the first places to start when looking to secure your code even more. The compiler will give the creator of the application feedback on what is missing if anything. Listening to these warnings is a great first step to making more secure code. |
| 1. Architect and Design for Security Policies | When designing an architecture for a system we always want to be prepared for the worst and plan for the best. We want to have many safety measures of backups and recalls to keep uptime high and we also want to implement best practices for keeping our software secure. |
| 1. Keep It Simple | We want to keep the application as simple as possible. Complex designs increase complexity for the system and often leads to more bugs and problems. As well as the easier a system the easier to maintain. |
| 1. Default Deny | When deciding who is allowed what access we will always use the default deny mythology. This means that every project starts with zero access and access is only handed out to those who are absolutely required to have it. |
| 1. Adhere to the Principle of Least Privilege | This relates to the concept of least access to an application. On the chance of the worst data breach, we only want our accounts to have very minimum access to anything. |
| 1. Sanitize Data Sent to Other Systems | This practice is to encrypt any data that is being sent to other systems just in case of the point of a breach on passage. If the worst happens and we are compromised in transit we want the hacker to have the least amount of data and we want it to be encrypted. |
| 1. Practice Defense in Depth | One encryption is never enough, we need two, or three. This concept of double and triple encryption is exactly what keeps people safe. Sometimes things get hacked and if there is only one wall of defense then that’s not enough. Sometimes input validation isn’t enough to stop a hacker. |
| 1. Use Effective Quality Assurance Techniques | Testing is always at the root of a good application. If you want to make sure your software is secure then test it. There’s multiple methods/ways to test and you should use them all. Don’t just stop at one way of testing. |
| 1. Adopt a Secure Coding Standard | Each language and application is different. Therefore, each one must have a set of rules to have as a standard of quality. Using code quality tools are also a major help in building applications. |

## 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-C++] | Naming Conventions Signed & Unsigned Integers |

| **Noncompliant Code** |
| --- |
| The Noncompliant code is incorrect uses of secure coding by having syntax errors for the 3rd line and for incorrectly computing the mathematical operation. Although it will compile and run it will not give correct results. |
| int target = -5; int num = 3;  target =- num; // Noncompliant; target = -3.  target =+ num; // Noncompliant; target = 3 |

| **Compliant Code** |
| --- |
| The compliant code is fixed with syntax that gives the incorrect answer to the problem trying to be solved. |
| int target = -5; int num = 3;  target = -num; // Compliant; target += num; |

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

|  |
| --- |
| **Principles(s):** Rule 5 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Major | Unlikely | Medium | High | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | https://rules.sonarsource.com/cpp/type/Bug?search=The%20use%20of%20operators%20pairs%20%28%3D%2B%29%20where%20the%20reversed%2C%20single%20operator%20was%20meant%20%28%2B%3D%29%20will%20compile%20and%20run%2C%20but%20not%20produce%20the%20expected%20results. | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | [STD-002-C++] | Data value is the value stored in variables and object throughout code. |

| **Noncompliant Code** |
| --- |
| In C++ the variables in memory need to be set free or you will use up all of your memory or worse leave it for hackers to play around in and find things they’re not supposed to find. Below is an example. |
| int fun() {  char\* name = (char \*) malloc (size);  if (!name) {  return 1;  }  return 0; // Noncompliant, memory pointed by "name" has not been released } |

| **Compliant Code** |
| --- |
| This code below is freeing up the variable from memory the correct way. |
| int fun() {  char\* name = (char \*) malloc (size);  if (!name) {  return 1;  }  free(name);  return 0; } |

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

|  |
| --- |
| **Principles(s):** Rule 1 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Blocker | Semi-Likely | Medium-High | High | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | https://rules.sonarsource.com/cpp/RSPEC-1232?search=Free%20memory | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | [STD-003-C++] | Strings are some of the most used variables. |

| **Noncompliant Code** |
| --- |
| This string below is concatenating of string literals with different encodings. Which isn’t going to be supported much longer in c++ |
| wchar\_t n\_array[] = "Hello" L"World"; // Noncompliant wchar\_t w\_array[] = L"Hello" "World"; // Noncompliant |

| **Compliant Code** |
| --- |
| Below is a correct version of combining strings that will last much longer. |
| char\_t n\_array[] = "Hello" "World"; // Compliant wchar\_t w\_array[] = L"Hello" L"World"; // Compliant |

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

|  |
| --- |
| Principles(s): Rule 5 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Major | Unlikely | Small | Major | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | https://rules.sonarsource.com/cpp/RSPEC-817 | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | [STD-004-C++] | Injection is when a user implements code into a project that wasn’t supposed to be there. |

| **Noncompliant Code** |
| --- |
| Below is an example of an overflow that could be used wrongly to inject code into SQL database |
| char buffer[10]; scanf("%s", buffer); // Noncompliant - will overflow when a word longer than 9 characters is entered |

| **Compliant Code** |
| --- |
| Below is the compliant code that will not overflow |
| char buffer[10]; scanf("%9s", buffer); // Compliant - will not overflow |

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

|  |
| --- |
| **Principles(s):** RULE 5 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Major | Likely | High | Major | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | Sonarcloud can check for overflow | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | [STD-005-C++] | Memory leaks are some of the most common data leak hacks |

| **Noncompliant Code** |
| --- |
| Only in c++ 17 and beyond, using char for negate byte is destined for errors since char allows for mathematical operations. |
| void handleFirstByte(char\* byte); void f(int\* i) {  char\* c = reinterpret\_cast<char\*>(i); // Noncompliant  handleFirstByte(c); } unsigned char negate(unsigned char byte) {  return ~byte; // Noncompliant } |

| **Compliant Code** |
| --- |
| This new format of handling this code is much more secure by not having gaps for injected expressions. |
| void handleFirstByte(std::byte\* byte); void f(int\* i) {  std::byte\* byte = reinterpret\_cast<std::byte\*>(i); // Compliant  handleFirstByte(byte); } std::byte negate(std::byte byte) {  return ~byte; // Compliant } |

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

|  |
| --- |
| **Principles(s):** Rule 6 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Major | Likely | High | Major | 5 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | https://rules.sonarsource.com/cpp/RSPEC-6022 | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | [STD-006-C++] | Method to for standard equals test |

| **Noncompliant Code** |
| --- |
| The code below is the worst example of how to use assert when you’re programming. |
| assert(this.isTrue == this.isTrue) //noncompliant |

| **Compliant Code** |
| --- |
| We want to test our code in this way to see if what we set is equal to what we expect. |
| assert(myObject.isTrue == True) //compliant |

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

|  |
| --- |
| **Principles(s):** Rule 8 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Medium | Low | Low | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | In SonarCloud/Qube you can make your own rules to ensure that this kind of testing doesn’t happen. | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | [STD-007-C++] | Exception handling is when you expect and catch errors within a program. |

| **Noncompliant Code** |
| --- |
| Below is an example of using a pointer as an exceptions which is a big no and something we never want to do. |
| class E { /\* Implementation \*/}; E globalException; void fn ( int i ){  if ( i > 10 ) {  throw ( &globalException); // Noncompliant, the catch is supposed not to delete the pointer  }  else {  throw (new E ); // Noncompliant, the catch is supposed to delete the pointer  } } |

| **Compliant Code** |
| --- |
| Here’s the compliant code that doesn’t use a pointer to the code. |
| class E { /\* Implementation \*/}; E globalException; void fn ( int i ){  if ( i > 10 ) {  throw ( globalException); // Throws a copy of the global variable } else {  throw (E{} ); // Throws a new object  } } |

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

|  |
| --- |
| **Principles(s):** Rule 8 |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | https://rules.sonarsource.com/cpp/RSPEC-1035 | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Naming Convention of Files | [STD-008-C++] | Every language has keywords that should never be used otherwise it will break the program. |

| **Noncompliant Code** |
| --- |
| Include\_next is specific to gcc and if ran on certain machines can cause complier issues |
| #include\_next "foo.h" // Noncompliant |

| **Compliant Code** |
| --- |
| For things that may be clashing with system words be very specific |
| #include "/usr/local/include/foo.h" |

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

|  |
| --- |
| **Principles(s):** Rule 1 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Minor | Unlikely | Low | Minor | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | https://rules.sonarsource.com/cpp/RSPEC-3730 | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Class protection | [STD-009-C++] | Child classes can see protected members but not private. When using a final class there is no protected members only private. |

| **Noncompliant Code** |
| --- |
| The code below is using protected instead of private which isn’t needed since it’s a final class |
| class C final { protected: // Noncompliant  void fun(); }; |

| **Compliant Code** |
| --- |
| Changing protected to private fixes this issue |
| class C final { private: // compliant  void fun(); }; |

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

|  |
| --- |
| Principles(s): Rule 2 |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | https://rules.sonarsource.com/cpp/RSPEC-2156 | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| No wasted code | [STD-010-C++] | Wasted code takes up compile/memory space that if ever leaked could expose thigs we don’t want. This goes back to our rule about least access. |

| **Noncompliant Code** |
| --- |
| Below shows dead code past a return statement. This code will never be used in any situation. |
| int fun(int a) {  int i = 10;  return i + a; // Noncompliant  i++; // dead code } |

| **Compliant Code** |
| --- |
| As best practice the dead code has been removed. |
| int fun(int a) {  int i = 10;  return i + a;  } |

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

|  |
| --- |
| **Principles(s):** Rule 2 |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Major | Low | Low | Major | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarCloud | 9.0 | https://rules.sonarsource.com/cpp/RSPEC-1763 | SonarCloud does static testing to review all code to make sure nothing is breaking any rules set above. |

## 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.

Using a CI/CD tool like Jenkins or GitLab/GitHub will allow us to add integration for our new code quality tool, SonarCloud. This will check out code in and out of production. On initial setup it will scan all code and check to make sure code passes all quality gates. After that each time new code is introduced into the repo it will be scanned and checked to pass quality assurance. This is mainly in the build phase but will stop all other phases after build phase if fails. It will also help devs test their code.

## 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 | Major | Unlikely | Medium | High | 2 |
| STD-002-CPP | Blocker | Semi-Likely | Medium-High | High | 5 |
| STD-003-CPP | Major | Unlikely | Small | Major | 1 |
| STD-004-CPP | Major | Likely | High | Major | 5 |
| STD-005-CPP | Major | Likely | High | Major | 5 |
| STD-006-CPP | Low | Medium | Low | Low | 1 |
| STD-007-CPP | Critical | Unlikely | Medium | High | 3 |
| STD-008-CPP | Minor | Unlikely | Low | Minor | 1 |
| STD-009-CPP | Minor | Unlikely | Low | Minor | 2 |
| STD-010-CPP | Major | Low | Low | Major | 2 |

## 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 works by changing data using some algorithm that only keyholders can use to decrypt and see the data. Since encryption at rest is just data not in motion then we usually need extra protection around this data to avoid getting the data stolen. |
| Encryption at flight | Encryption at flight is when you encrypt data right before sending or sometimes as you’re sending it. Sometimes data isn’t encrypted at all until sent. |
| Encryption in use | Out of the options above they have more of a difference depending on the situation. Encryption in use is when data is encrypted all the time. Either on your own devices or in the cloud or in transit the data is always encypted. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication takes into account all user logins, changes to the database, addition of new users, user level of access, and files accessed by users. None of these things can happen without proper authentication. If unauthenticated the user cannot do anything. |
| Authorization | Authentication depends on who you are but authorization looks at what you can do. In companies not everyone has the same job. So not everyone should see the same things. We don’t give the intern full super admin access. It’s just a bad idea on all parts. |
| Accounting | Accounting is more along the lines of monitoring what is going on. Let’s say we have a large file system that 50 users are using at once and suddenly things were stolen. With accounting we can track who did what and log our activity to be more secure. |

**\***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

Principals (Using Carnegie defined principles from rubric) Link [here](https://wiki.sei.cmu.edu/confluence/display/cplusplus/STR53-CPP.+Range+check+element+access)

STD-001-CPP = Rule 5(STR)

STD-002-CPP = Rule 1(DCL)

STD-003-CPP = Rule 5(STR)

STD-004-CPP = Rule 5(STR)

STD-005-CPP = Rule 6(MEM)

STD-006-CPP = Rule 8(ERR)

STD-007-CPP = Rule 8(ERR)

STD-008-CPP = Rule 1(DCL)

STD-009-CPP = Rule 2(EXP)

STD-010-CPP = Rule 2(EXP)

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 |