**Green Pace Developer: Security Policy Guide**



# Green Pace Secure Development Policy

# By Thomas Fiske

# 6/7/2023

## 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 | The code written must validate input from users and other places of it will leave the system open to attacks such as SQL injection. |
| 1. Heed Compiler Warnings | Listen to the compiler when it finds errors and fix them. |
| 1. Architect and Design for Security Policies | The design of your system should be based around security. |
| 1. Keep It Simple | Do not overcomplicate code. It makes it hard to read and maintain among other things. |
| 1. Default Deny | The default into the system should be to deny users to not accidently let someone in with the wrong credentials. |
| 1. Adhere to the Principle of Least Privilege | The less amount of people that have access to sensitive data, the more protected it will be. Do not allow a lot of people to have privileges that they do not need to have. |
| 1. Sanitize Data Sent to Other Systems | When sending data, make sure that there is no malware or other harmful data attached to it. |
| 1. Practice Defense in Depth | Practice having layer after layer of security in order to safeguard the best from attacks. |
| 1. Use Effective Quality Assurance Techniques | Use testing and other quality assurance techniques to make sure that the system is running correctly / functioning correctly. |
| 1. Adopt a Secure Coding Standard | Adopt a standard of coding that will be secure and efficient for your system. |

### 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** | DCL51-CPP | Do not declare or define a reserved identifier |

| **Noncompliant Code** |
| --- |
| This code using header files names that could clash with already existing files because of their beginning \_ |
| #ifndef \_myHeader  #define \_myHeader  #endif \_myHeader |

| **Compliant Code** |
| --- |
| This code changed the names of the headers to be compliant by removing the \_ At the beginning. |
| #ifndef myHeader  #define myHeader  #endif myHeader |
|  |

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

| **Principles(s):** Use effective quality assurance techniques, this error should be taken care of during QA. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Reserved-identifier | Partially checked |
| Helix QAC | 2023.1 | C++5003 |  |
| RuleChecker | 22.10 | Reserved-identifier | Partially checked |
| PVS-Studio | 7.25 | V1059 |  |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | INT31-C | Ensure that integer conversions do not result in lost or misinterpreted data |

| **Noncompliant Code** |
| --- |
| This code sets up the operation within an IF/ELSE statement so that if it fails it does not result in an error/loss of data. |
| Void function(void){  Unsigned long int Tf = ULONGMAX;  Signed char sc;  Sc = (signed char)tf;  } |

| **Compliant Code** |
| --- |
| This code sets up the operation within an IF/ELSE statement so that if it fails it does not result in an error/loss of data. |
| Void function(void){  Unsigned long int Tf = ULONGMAX;  Signed char sc;  If (tf <= SCHAR\_MAX) {  Sc = (signed char)tf;  }  Else {  Cout << “Error, function did not work”  }  } |

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

| **Principles(s):** Heed compiler warnings. This code will show compiler warnings and should be fixed before running. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 |  | Supported via MISRA C:2012 Rules 10.1, 10.3, 10.4, 10.6, 10.7 |
| Compass/ROSE |  |  | Can detect violations of this rule. However, false warnings may be raised if limits.h is included. |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Klocwork | 2023.1 | PORTING.CAST.SIZE |  |

#### Coding Standard 3

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

| **Noncompliant Code** |
| --- |
| In this example, the variable string is a string literal which cannot be modified which is what the second statement tries to do, causing an error. |
| Char \*string = “string literal”;  Str[0] = “String”; |

| **Compliant Code** |
| --- |
| This fixes the code by changing the variable string into an array so the second statement can proceed. |
| Char string[] = “string literal”;  Str[0] = “String”; |

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

| **Principles(s):** Keep it simple. Making sure that strings are declared and used correctly is a way to keep code simple and error free. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | String-literal-modification  Write-to-string-literal | Fully checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC-STR30 | Fully implemented |
| Coverity | 2017.07 | PW | Deprecates conversion from string literal to “char \*” |
| Compass/ROSE |  |  | Can detect simple violations of this rule |

#### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | FIO30-C | Exclude user input from format string |

| **Noncompliant Code** |
| --- |
| This code does not check the query that the user is putting as the password to search the database. This could leave the system open to SQL injection attacks. |
| Void inputPassword (userInput){  Cout << “input password”;  String input = “”;  Cin >> input;  If (input == userpassword){  allowAccess()  }    } |

| **Compliant Code** |
| --- |
| This code adds a function that checks the query that the user entered to make sure that there are no added strings. |
| Void inputPassword (userInput){  Cout << “input password”;  String input = “”;  Cin >> input;  checkQuery(input);  If (input == userpassword){  allowAccess()  }    } |

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

| **Principles(s):** Validate input data. This is a textbook example of validating input data to prevent SQL attacks. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 |  | Supported via stubbing/taint analysis |
| Axivion Bauhaus Suite | 7.2.0 | CertC-FIO30 | Partially implemented |
| CodeSonar | 7.3p0 | IO.INJ.FMT  MISC.FMT | Format string injection  Format string |
| Compass/ROSE |  |  |  |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | MEM30-C | Do not access freed memory |

| **Noncompliant Code** |
| --- |
| In this case p->next will read data that has already been freed. |
| 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** |
| --- |
| This changes the code so that the data has not already been freed when p->next is called. |
| Struct node{  Int value;  Struct node \*next;  };  Void free\_list(struct node \*head){  For (struct node \*p = head; p != NULL; p = 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. This is a code error that should be caught in the QA part of development. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | Dangling\_pointer\_use | Supported  Astree reports all accesses to freed allocated memory |
| Axivion Bauhaus Suite | 7.2.0 | CertC-MEM30 | Detected memory accesses after its deallocation and double memory deallocations |
| CompassROSE |  |  |  |
| CodeSonar | 7.3p0 | ALLOC.UAF | Use after free |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | EXP46-C | Do not use a bitwise operator with a Boolean-like operand. |

| **Noncompliant Code** |
| --- |
| This IF statement is true but it is only using one & instead of two which will cause an error. |
| If ((100 = 100) & (10 = 10) {  doSomething();  } |

| **Compliant Code** |
| --- |
| This code is fixed so that it is using the correct terminology. |
| If ((100 = 100) && (10 = 10) {  doSomething();  } |

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

| **Principles(s):** Adopt a secure coding standard. This is a syntax error that should be caught by the developer if they are following their secure coding standard. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 23.04 | Inappropriate-bool | Supported indirectly via MISRA C:2012 Rule 10.1 |
| Axivion | 7.2.0 | CertC-EXP46 |  |
| CodeSonar | 7.3p0 | LANG.TYPE.IOT | Inappropriate operand type |
| Cppcheck | 1.66 | Cert.py | Detected by the addon cert.py |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | ERR51-CPP | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| This code will fail because the function is trying an incorrect statement and does not handle the exception. |
| Void functionTry(){  Int var = 8;  Double var2 = 10.00  Var = v2;  }  Int main(){  functionTry();  } |
|  |

| **Compliant Code** |
| --- |
| This code will catch the exception that is thrown and tell the developer/user. |
| Void functionTry(){  }  Int main(){  Try{  functionTry();  }  Catch (Error) {  Cout << “Error with functionTry”;  } |

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

| **Principles(s):** Adopt a secure coding standard. Having a standard with how to deal with exceptions for each system makes it more uniform and allows exceptions to be handled correctly. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astree | 22.10 | Main-function-catch-all  Early-catch-all | Partially checked |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR51 |  |
| CodeSonar | 7.3p0 | LANG.STRUCT.UCTCH | Unreachable catch |
| Helix QAC | 2023.1 | C++4035, C++4036, C++4037 |  |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Error Detect | ERR62-CPP | Detect errors when converting a string to a number |

| **Noncompliant Code** |
| --- |
| This code does not catch the error that is thrown from trying to convert a string to a number. |
| Void function() {  Int j;  String k;  K = j;  } |

| **Compliant Code** |
| --- |
| This code adds a try and catch statement to catch the errors. |
| Void function() {  Int j;  String k;  Try {  J = k;  }  Catch (error e) {  Cout << “error at : “ << e;  }  } |

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

| **Principles(s):** Heed compiler warnings. This is something that should show a warning in the compiler which should be heeded and fixed. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-ERR62 |  |
| Clang | 3.9 | Cert-err34-c | Checked by clang-tidy; only identifies use of unsafe C Standard Library functions corresponding to ERR34-C |
| Helix QAC | 2023.1 | C++3161 |  |
| Klocwork | 2023.1 | CERT.ERR.CONV.STR\_TO\_NUM |  |

#### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Declaration | DCL60-CPP | Obey the one-definition rule |

| **Noncompliant Code** |
| --- |
| This code declares the same name for two different data types in two files used by the project. |
| //File 1.cpp  Struct structure {  Int var;  };  //file 2.cpp  Class structure {  Int var;  }; |

| **Compliant Code** |
| --- |
| This code fixes the double declaration problem and includes the file instead. |
| //File 1.cpp  Struct structure {  Int var;  };  //file 2.cpp  #include File 1.cpp |

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

| **Principles(s):** Adopt a secure coding standard. Following the one-definition rule is a good example of having a good secure coding standard that helps the developer develop secure code. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | CertC++-DCL60 |  |
| CodeSonar | 7.3p0 | LANG.STRUCT.DEF.FDH  LANG.STRUCT.DEF.ODH | Function defined in header file  Object defined in header file |
| Helix QAC | 2023.1 | C++1067, C++1509, C++1510 |  |
| LDRA tool suite | 9.7.1 | 286S, 287 S | Fully implemented |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| Constraints | MSC40-C | Do not violate constraints |

| **Noncompliant Code** |
| --- |
| This code has a constraint on var that it has to be static and then violates that. |
| Static int var = 20;  Extern inline void func(int var2){  Int var 3 = var \* var2  } |

| **Compliant Code** |
| --- |
| The static declaration was removed to allow the function to work properly. |
| int var = 20;  Extern inline void func(int var2){  Int var 3 = var \* var2  } |

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

| **Principles(s):** Architect and design for security policies. This applies because a good design will help avoid issues like violating constraints. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Klockwork | 2023.1 | MISRA.FUNC.STATIC.REDECL |  |
| Parasoft C/C++test | 2022.2 | CERT\_C-MSC40-a | An inline definition of a function with external linkage shall not contain definitions and uses of static objects |
| Polyspace Bug Finder | R2023a | CERT C: Rule MSC40-C | Checks for inline constraint not respected (rule partially covered) |
| LDRA tool suite | 9.7.1 | 21 S, 145 S, 323 S, 345 S, 387 S, 404 S, 481 S, 580 S, 612 S, 615 S, 646 S |  |

### 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.

Automation can be woven into the DevOps process by using the tools that have been listed in this policy. These tools will do thousands of work hours’ worth of time in a fraction of the time. Automation allows the developer/tester to do the work of multiple developers/testers by doing the work of checking the system for these errors. These tools should be used in the verify and test part of the DevOps process because this will allow easy transition into using these tools and improve this state of the process.

### 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 |
| DCL51-CPP | Low | Unlikely | Low | P3 | L3 |
| DCL60-CPP | Unlikely | High | P3 | L3 | Unlikely |
| EXP46-C | Low | Likely | Low | P9 | L2 |
| INT31-C | High | Probable | High | P6 | L2 |
| STR30-C | Low | Likely | Low | P9 | L2 |
| MEM30-C | High | Likely | Medium | P18 | L1 |
| FIO30-C | High | Likely | Medium | P18 | L1 |
| ERR51-CPP | Low | Probable | Medium | P4 | L3 |
| ERR62-CPP | Medium | Unlikely | Medium | P4 | L3 |
| MSC40-C | Low | Unlikely | Medium | P2 | 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 | This means that the data is encrypted where it is stored. It applies because it is a security measure to further increase security. |
| Encryption at flight | Encryption during the transmitting of data, this is extremely important as connections can be intercepted. |
| Encryption in use | Encryption when a piece of data is being used. This means that it can only be viewed/decrypted by the person using it if they have the key. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | This means to use credentials to check access. It applies because not everyone should have access and that needs to be authenticated. This is usually done through a user login. |
| Authorization | This means to grant access. A person should only have the amount of access to files that is absolutely necessary for them and nothing higher. This will help prevent confidential information from leaking. |
| Accounting | This means to keep track of what is going on inside of the system/application. The user moves need to be accounted for so that if there is a breach, it can be identified how/where it happened. |

**\***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

# DCL51-CPP

1. Heed Compiler Warnings
2. Architect and Design for Security policies
3. Keep it simple
4. Use Effective Quality Assurance Techniques

# DCL60-CPP

1. Heed Compiler Warnings
2. Use Effective Quality Assurance Techniques
3. Adopt A Secure Coding Standard

# EXP46-C

1. Heed Complier Warnings
2. Architect and Design for Security Policies
3. Use Effective Quality Assurance Techniques
4. Adopt A Secure Coding Standard

# INT31-C

1. Heed Complier Warnings
2. Architect and Design for Security Policies
3. Use Effective Quality Assurance Techniques
4. Adopt A Secure Coding Standard

# STR30-C

* + 1. Use Effective Quality Assurance Techniques
    2. Heed Compiler Warnings
    3. Adopt A Secure Coding Standard
    4. Architect and Design for Security Policies

# MEM30-C

* + 1. Use Effective Quality Assurance Techniques
    2. Heed Compiler Warnings
    3. Adopt A Secure Coding Standard
    4. Architect and Design for Security Policies

# FIO30-C

* + 1. Default Deny
    2. Adopt a Secure Coding Standard
    3. Architect and Design for Security Policies
    4. Practice Defense in Depth

# ERR51-CPP

* + 1. Adopt a Secure Coding Standard
    2. Architect and Design for Security Policies
    3. Practice Defense in Depth
    4. Heed Compiler Warnings
    5. Use Effective Quality Assurance Techniques

# ERR62-CPP

* + 1. Use Effective Quality Assurance Techniques
    2. Heed Compiler Warnings
    3. Adopt A Secure Coding Standard
    4. Architect and Design for Security Policies

# MSC40-C

* + 1. Use Effective Quality Assurance Techniques
    2. Heed Compiler Warnings
    3. Adopt A Secure Coding Standard
    4. Architect and Design for Security Policies

A lot of these coding standards can be met with four of the security principals. Use Effective Quality Assurance Techniques, Heed Compiler Warnings, Adopt a Secure Coding Standard, and Architect and Design for Security Policies are used a lot by these different coding standards because they are very good principals to stick to when secure coding. Following these principals will help the developer continue to maintain a high level of secure code and will also allow them to follow the secure coding rules. It is also important though to follow the rest of the coding principals because they all together help prevent attacks and hacks by malicious actors.

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 | 6/11/2023 | Security Policy update | Thomas Fiske |  |
| [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 |