**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)

[Data Type Coding Standard 4](#_Toc52464059)

[Data Value Coding Standard 5](#_Toc52464060)

String Correctness [Coding Standard 6](#_Toc52464061)

SQL Injection [Coding Standard 7](#_Toc52464062)

Memory Protection [Coding Standard 8](#_Toc52464063)

Assertions [Coding Standard 9](#_Toc52464064)

Exceptions [Coding Standard 10](#_Toc52464065)

Object Oriented Programming [Coding Standard 11](#_Toc52464066)

Declarations and Initialization [Coding Standard 13](#_Toc52464067)

Miscellaneous [Coding Standard 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 | We must assume that any input is a malicious attack and therefore we cannot rely on the client-side for validation, we must strictly use canonicalization and normalization with the user input. After it goes through this process, we must ensure that the input is sanitized before being validated; by doing this we can limit the attacks that get through and hit our systems. |
| 1. Heed Compiler Warnings | To keep our code clean and professional, we need to ensure that we are checking our compiler warnings before pushing any new code changes to the central location which our projects are being held in for testing and pull requests. If we are submitting code that has compiler warnings, then supervisors/team leaders will have to discuss with that team member as to why they pushed the code without talking to them first. |
| 1. Architect and Design for Security Policies | The important thing we must do is ensure that all user information is encrypted and the databases that hold this information are configured correctly so the only way users can access it is when they are logging in and the program is able to confirm it is them by validating their input. |
| 1. Keep It Simple | Keeping our code simple and limiting what functions it does to what the customer needs will help us to return to the project if an employee leaves as well as limiting our attack surface. By building a project that is more complex means we have more code to cover and protect when we could have done the same project in less code. |
| 1. Default Deny | We must deny any default information being used especially when it comes to any network settings because these are called “default settings” for a reason in which if you use them then the attacker can get through easier compared to changing settings specific to your needs. |
| 1. Adhere to the Principle of Least Privilege | We do not need to give people access that they do need to have access to, a user working in accounting does not need admin access to their own computer or the network itself however if someone works in IT or cyber security section then they need this information. |
| 1. Sanitize Data Sent to Other Systems | Doing this will ensure we are not accidentally sending data to other systems that they do not need or if we do get malicious code input into our system then we are not spreading that to our clients. This also ensures that users are not sending data to outside systems that we do not need to be sending and if we are then we are encrypting it before it goes out. |
| 1. Practice Defense in Depth | A plan to ensure that we are protecting our systems from outsider and insider threats while also ensuring that we are not keeping certified users from accessing programs or data that they need to do their jobs. |
| 1. Use Effective Quality Assurance Techniques | We must test all code before pushing the changes to our central location of the project in not just ensuring that the code works but the testing teams or cyber security teams need to see if the code as any vulnerabilities that can be exploited. |
| 1. Adopt a Secure Coding Standard | Each team needs to adopt a secure coding standard dependent on their relation to certain tasks, if they do this then we are able to ensure that our projects are better secure. |

### 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.

#### Data Type Coding Standard

| **Coding Standard** | **Label** | **Do Not Read Uninitialized Memory** |
| --- | --- | --- |
| **Data Type** | [STD-053-CPP] | This can cause the program to behave in an undefined manor and therefore we have no standard on how the program should react when it reaches this line of code or function. |

| **Noncompliant Code** |
| --- |
| In this line of code, we have a variable that has no definition, and it is being called upon to be printed. |
| #include <iostream>  void d(){  int h;  std :: cout << h;  } |

| **Compliant Code** |
| --- |
| In this line of code, we used the same code from above however we gave the variable *h* a value which causes it to be initialized. |
| #include <iostream>  void d(){  int h = 0;  std :: cout << h;  } |

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

| **Principles(s):** Economy of Mechanism – By keeping the program as simple as possible, we are not trying to use something that is not already in use. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wuninitialized  Clang-analyzer-core.UndefinedBinaryOperatorResult | Does not catch all instances of this rule, such as uninitialized values read from heap-allocated memory. |
| LDRA Tool Suite | 9.7.1 | **53 D, 69 D, 631 S, 652 S** | Partially implemented |
| Parasoft Institute |  |  | Runtime detection |
| Polyspace Bug Finder | R2021a | CERT C++: EXP53-CPP | Checks for:   * Non-initialized variable * Non-initialized point   Rule partially covered. |

#### Data Value Coding Standard

| **Coding Standard** | **Label** | **Do Not Cast To An Out-of-Range Enumeration Value** |
| --- | --- | --- |
| **Data Value** | [STD-050-CPP] | We must check to ensure the value can be represented before it is casted otherwise the value may not truly be given. |

| **Noncompliant Code** |
| --- |
| Here we have the code is trying to cast the variable before it is ensuring the value can be represented. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    EnumType enumVar = static\_cast<EnumType>(intVar);      if (enumVar < First || enumVar > Third) {      // Handle error    }  } |

| **Compliant Code** |
| --- |
| This code will ensure the variable is able to be handled before casting and thus it is ensuring that we are not causing any errors. |
| enum EnumType {    First,    Second,    Third  };    void f(**int** intVar) {    if (intVar < First || intVar > Third) {      // Handle error    }    EnumType enumVar = static\_cast<EnumType>(intVar);  } |

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

| **Principles(s):** Complete mediation |
| --- |

**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++-INT50** |  |
| Parasoft C/C++test | 2021.2 | **C++3013** | An expression with enum underlying type shall only have values corresponding to the enumerators of the enumeration |
| PRQA QA-C++ | 4.4 | **3013** |  |
| PVS-Studio | 7.15 | **V1016** |  |

#### String Correctness Coding Standard

| **Coding Standard** | **Label** | Pass An Object Of The Correct Type To va\_start |
| --- | --- | --- |
| **String Correctness** | [STD-058-CPP] | This can cause a default argument promotion and therefore causes undefined behavior from the program. |

| **Noncompliant Code** |
| --- |
| This causes a default argument promotion with the float implementation. |
| #include <cstdarg>  extern “B” void f(float a,…){  va\_list list;  va\_start (list, a);  va\_end(list);  } |

| **Compliant Code** |
| --- |
| By using double instead of float, we are able to keep the default argument promotion from happening and that is due to double having twelve decimal places as opposed to float’s four. |
| #include <cstdarg>  extern “B” void f(double a,…){  va\_list list;  va\_start (list, a);  va\_end(list);  } |

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

| **Principles(s):** least common mechanism |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Clang | 3.9 | -Wvarargs | Does not catch the violation in the third noncompliant code example (it is conditionally supported by Clang) |
| Helix QAC | 2021.2 | **C++3852, C++3853** |  |
| Parasoft C/C++test | 2021.1 | **CERT\_CPP-EXP58-a** | Use macros for variable arguments correctly |
| Polyspace Bug Finder | R2021a | CERT C++: EXP53-CPP | Checks for incorrect data types for second argument va\_start (rule fully covered) |

#### SQL Injection Coding Standard

| **Coding Standard** | **Label** | Sanitize Data Passed To Complex Subsystems |
| --- | --- | --- |
| **SQL Injection** | [STD-002-C] | By understanding what data is being passed to certain subsystems and the capabilities of those subsystems we able to ensure that the data going to them are not allowing bad actors to enter with bad inputs. |

| **Noncompliant Code** |
| --- |
| This does not set up any bad characters or check to ensure that characters are being checked before being sent to the subsystem. |
| sprintf(buffer, “/bin/mail %s < /tmp/email”, addr);  springf(buffer); |

| **Compliant Code** |
| --- |
| Here the code actually takes the user input and ensures that the input is good before it is sent to the subsystem. |
| static char good\_chars[]=”abcdefghijklmnopqrstuvwxyz”  “ABCDEFGHIJKLMNOPQRSTUVWXYZ”  “1234567890\_-.@”;  char user\_data[]= “Bad char 1:} Bad char 2:{“;  char \*cp = user\_data;  const char \*end = user\_data + strlen( user\_data);  for (cp += strspn(cp, good\_chars); cp != end; cp += strspn(cp, good\_chars)){  \*cp = ‘\_’;  } |

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

| **Principles(s):** least privilege; separation of privilege |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 6.5 | **TAINTED\_STRING** | Fully implemented |
| Klocwork | 2021.1 | **NNTS.TAINTED**  **SV.TAINTED.INJECTION** |  |
| LDRA tool suite | 9.7.1 | **108 D, 109 D** | Partially implemented |
| Parasoft C/C++test | 2021.1 | **CERT\_C-STR02-a**  **CERT\_C-STR02-b**  **CERT\_C-STR02-c** | Protect against command injection  Protect against file name injection  Protect against SQL injection |

#### Memory Protection Coding Standard

| **Coding Standard** | **Label** | Do Not Access Freed Memory |
| --- | --- | --- |
| **Memory Protection** | [STD-050-CPP] | Once memory is deallocated then the remaining points are dangling and accessing those can result in exploitable vulnerabilities due to the data there can be changed unexpectedly which means an outsider can change it to allow them access to other systems with malicious code. |

| **Noncompliant Code** |
| --- |
| In this code we have deleted *s* but then we called upon it once more to move it into *f* and therefore it creates a possible vulnerability. |
| #include <new>  struct S {  void f();  };  void g() noexcept(false){  S \*s = new S;  delete s;  s->f();  } |

| **Compliant Code** |
| --- |
| Here we are moving *s* into *f* and then we delete *s* which means we are no longer touching *s* after deallocating it. |
| #include <new>  struct S {  void f();  };  void g() noexcept(false){  S \*s = new S;  s->f();  delete s;  } |

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

| **Principles(s):** least common mechanism; economy of mechanism |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++MEM50** |  |
| CodeSonar | 6.1p0 | **ALLOC.UAF** | Use after free |
| Helix QAC | 2021.2 | **C++4303, C++4304** |  |
| Polyspace Bug Finder | R2021a | CERT C++:MEM-50CPP | Checks for:   * Pointer access out of bounds * Deallocation of previously deallocated pointer * Use of previously freed pointer   Rule partially covered |

#### Assertions Coding Standard

| **Coding Standard** | **Label** | Use a Static Assertion to Test the Value of a Constant Expression |
| --- | --- | --- |
| **Assertions** | [STD-003-C] | This allows us to check for software defects that could result in vulnerabilities and is suitable for server programs or embedded systems as opposed to the runtime assert function. |

| **Noncompliant Code** |
| --- |
| This will only run if the function is called upon and is therefore too far away (most likely) from the structure it references. |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  int func(void) {  assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int));  } |

| **Compliant Code** |
| --- |
| By not requiring to be inside a function, this can run right along the structure that it references with little distance. |
| #include <assert.h>  struct timer {  unsigned char MODE;  unsigned int DATA;  unsigned int COUNT;  };  static\_assert(sizeof(struct timer) == sizeof(unsigned char) + sizeof(unsigned int) + sizeof(unsigned int),  “Structure must not have padding”); |

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

| **Principles(s):** open design |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC-DCL03** |  |
| Clang | 3.9 | misc-static-assert | Checked by clang-tidy |
| ÉCLAIR | 1.2 | **CC2.DCL03** | Fully implemented |
| LDRA tool suite | 9.7.1 | **44 S** | Fully implemented |

#### Exceptions Coding Standard

| **Coding Standard** | **Label** | Handle All Exceptions |
| --- | --- | --- |
| **Exceptions** | [STD-051-CPP] | We must handle all exceptions in order to prevent from denial-of-service attacks, if an exception does not have a handle then it can cause the program to terminate and if there is a mass issue of these then it can cause the entire system to crash. |

| **Noncompliant Code** |
| --- |
| This will not handle the exception and therefore it will cause the program to crash entirely. |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  f();  } |

| **Compliant Code** |
| --- |
| By handling this error, the program does not crash and therefore it can tell the user there is an error with they put in and to try again. |
| void throwing\_func() noexcept(false);  void f() {  throwing\_func();  }  int main() {  try {  f();  } catch (..){  //handles error  }  } |

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

| **Principles(s):** open design; economy of mechanism |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-ERR51** |  |
| LDRA tool suite | 9.7.1 | **527 S** | Partially implemented |
| Polyspace Bug Finder | R2021a | Cert C++:ERR51-CPP | Checks for unhandled exceptions (rule partially covered) |
| PRQA QA-C++ | 4.4 | **4035, 4036, 4037** |  |

#### Object Oriented Programming Coding Standard

| **Coding Standard** | **Label** | Do Not Delete a Polymorphic Object Without A Virtual Destructor |
| --- | --- | --- |
| **Object Oriented Programming** | [STD-052-CPP] | Without using a virtual destructor, we could be causing the memory that was being used by an object or instance to remain being used but with it that memory is freed when that object or instance is deleted. |

| **Noncompliant Code** |
| --- |
| Base does not have a destructor declared as virtual even though the presence of other virtual functions |
| struct Base {  virtual void f();  };  struct Derived : Base {};  void f() {  Base \*b = new Derived();  //…  delete b;  } |

| **Compliant Code** |
| --- |
| We now have base with a virtual destructor which will now free up the memory that was used. |
| struct Base {  virtual ~Base() = default;  virtual void f();  };  struct Derived : Base {};  void f() {  Base \*b = new Derived();  //…  delete b;  } |

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

| **Principles(s):** least common mechanism; open design |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CERTC++-OOP52** |  |
| Clang | 3.9 | -Wdelete-non-virtual-dtor |  |
| LDRA tool suite | 9.7.1 | **303 S** | Partially implemented |
| PRQA QA-C++ | 4.4 | **3402, 3403, 3404** |  |

#### Declarations and Initialization Coding Standard

| **Coding Standard** | **Label** | Do Not Modify the Standard Namespaces |
| --- | --- | --- |
| **Declarations and Initialization** | [STD-058-CPP] | By adding declarations into the standard namespace, we cause undefined behavior, and we should only do this under special circumstances. |

| **Noncompliant Code** |
| --- |
| We have added the declaration of *x* into the standard namespace and therefore the program will act in a way we did not intend for it to act. |
| namespace std {  int x;  } |

| **Compliant Code** |
| --- |
| By changing the namespace to *nonstd* we are able to add the declaration into a namespace without a reserved name, as the one above could cause collisions with other global identifiers. |
| namespace nonstd {  int x;  } |

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

| **Principles(s):** open design; least common mechanism |
| --- |

**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++-DCL58** |  |
| Klocwork | 2021.1 | **CERT.DCL.STD\_NS\_MODIFIED** |  |
| Polyspace Bug Finder | R2021a | CERT C++: DCL58-CPP | Checks for modification of standard namespaces (rule fully covered). |
| [Insert text.] | [Insert text.] | [Insert text.] | [Insert text.] |

#### Miscellaneous Coding Standard

| **Coding Standard** | **Label** | **Do Not Use std::rand() For Generating Pseudorandom Numbers** |
| --- | --- | --- |
| **Miscellaneous** | [STD-50-CPP] | The numbers provided by *std::rand()* can be predictive as they have a short cycle in which this command is used and therefore makes any numbers being guessed made easier to find for bad actors. |

| **Noncompliant Code** |
| --- |
| If a bad actor knows we are using this code for IDs then they can reverse engineer the code to find the IDs of all users or to find a specific user, once they have the ID it is just one step of them getting into our system. |
| #include <cstdlib>  #include <string>  void f() {  std::string id (“ID”);  id += std::to\_string(std::rand() % 10000);  } |

| **Compliant Code** |
| --- |
| By running our random number generation through various steps we can cause the ID #s to be completely random which makes figuring them out harder and the bad actor would have to access our system in a different fashion in order to find any specific ID. |
| #include <cstdlib>  #include <string>  void f() {  std::string id (“ID”);  std::uniform\_int\_distribution<int> distribution(0, 10000);  std::random\_device rd;  std::mt19937 engine (rd());  id += std::to\_string(distribution(engine));  } |

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

| **Principles(s):** open design |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Axivion Bauhaus Suite | 7.2.0 | **CertC++-MSC50** |  |
| Clang | 4.0 | cert-msc50-cpp | Checked by clang-tidy |
| ÉCLAIR | 1.2 | **CC2.MSC30** | Fully implemented |
| LDRA Tool Suite | 9.7.1 | **44 S** | Enhanced Enforcement |

### 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.



We are going to use automation tools to test our code, by doing this we are cutting down on testing time and we can determine during our DevOps process if there are any vulnerabilities that we need to address before pushing the changes to any repositories for that project. If there are any, we can assess them and plan our attack on fixing them, depending on their level of vulnerability; after doing so we can move on to building the code and then verify and test it before making that push.

### 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-053-CPP | High | Probable | Medium | High | 1 |
| STD-50-CPP | Medium | Unlikely | Medium | Low | 3 |
| STD-058-CPP | Medium | Unlikely | Medium | Low | 3 |
| STD-002-C | High | Likely | Medium | Medium | 1 |
| STD-050-CPP | High | Likely | Medium | Medium | 1 |
| STD-003-C | Low | Unlikely | High | Medium | 3 |
| STD-051-CPP | Low | Probably | Medium | Low | 3 |
| STD-052-CPP | Low | Likely | Low | Medium | 2 |
| STD-058-CPP | High | Unlikely | Medium | Medium | 2 |
| STD-050-CPP | Medium | Unlikely | Low | Medium | 2 |

### Create Policies for Encryption and Triple A

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 | For at rest, we will make sure that only people who need access to data that is not being moved or used are only accessing what they need and having logs who access it; by encrypting our data and controlling who has access to it we can limit our attack surface to ransomware. |
| Encryption at flight | Our policy for encryption in flight is to ensure that our data is being encrypted when we are transmitting our data, by doing this we are ensuring that if someone intercepts our data they will need the hash table in order to decrypt the data and they could not try to ransom it off to us. |
| Encryption in use | For encryption in use, we will be making sure the data is encrypted while being used so if someone this will limit anyone who can see inside of our computers but we will need to make sure that administrative passwords are controlled items to make sure that anyone who is “inside” our computers can not actually read the data. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | By controlling authentication and enabling multi-factor authentication protocols in all that we do, especially for when people are logging in new devices, then we can limit someone breaking through someone’s username/password combo and if anyone gets an email or notification of a new log in device then they are required to report it right away so their account can be locked until they are able to ensure their account and access is not breached. |
| Authorization | Users will need supervisors to approve them access under supervision of the cyber-defense team and the IT department before they are able to make any changes or take any equipment home. |
| Accounting | To monitor the user data being sent and received to ensure that they are using the equipment properly, if the user is using equipment outside their company duties then their access becomes limited and they have to report (with supervisor) to the IT department. |

**\***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 |  |
| 2.0 | 10/10/2021 | Complete Template | Daniel Holcomb |  |
| [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 |