**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.

## Ten Core Security Principles



| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | **Validate Input Data** is based around eliminating vulnerabilities such as SQL and command line injection attacks. The goal is to prevent an attacker from short circuiting an authentication loop to get access to privileged data in an application. This also extends to an API that receives and acts upon data in the request body. All untrusted data should be validated before acting upon it. |
| 1. Heed Compiler Warnings | **Compiler Warnings** can often tell you where your code may be leaking information. While the warnings in Visual Studio or most standard Integrated Development Environment (IDE) are usually sufficient, consider using static or dynamic analysis tools to get more in-depth and eliminate additional security flaws. |
| 1. Architect and Design for Security Policies | **Architect and Design for Security Policies** means that software should be designed for and incorporate security from initial inception, and not bolted on later. An example of this would be to utilize the least permissions possible, and potentially break up your application into subsystems if higher permissions are needed. |
| 1. Keep It Simple | **Keep It Simple** refers to how you design the application overall. If your application has less code and overall is less complex, the time it takes to test and validate during development is reduced. Also, configuration and use is more straightforward, and is less likely to encounter errors. Generally speaking, less code = less real-world issues. |
| 1. Default Deny | When designing the security model of an application, the author has a choice of designing the overall access model. **Default Deny** suggests that the security model should deny access in all cases, and only define the exclusions that allow access. This is opposite of designing an open system and defining exclusions to make it closed. The benefits of designing **Default Deny** is generally the code making up this module is simpler, and you have less of a chance of leaking data. |
| 1. Adhere to the Principle of Least Privilege | As mentioned above, an application should always **adhere to the principal of least privilege.** This pretty much states that code should execute with the lowest applicable permissions, and only gain permissions to execute at that lowest level (in the case where permissions need to be requested). Having too high of permissions allow the application to access resources it may not need, and an attacker could use this to overwhelm a system. |
| 1. Sanitize Data Sent to Other Systems | **Sanitize Data Sent to Other Systems** talks about how attackers can utilize data sent to a system to attack the system. An example of this is SQL injection attacks which if the passed query string is not protected, can cause authentication to fail and allow unfettered access. This principal also governs turning off and removing unused functionality, especially in commercial products as they may be attack vector to gain access to the system. |
| 1. Practice Defense in Depth | **Defense in Depth** utilizes multiple layers of defense to prevent a security flaw from becoming a full-fledged vulnerability. Additionally, having layers of defense can limit the damage in the case of a successful intrusion or exploit. An example of this would be to combine secure programming techniques in a secured environment to defend against someone gaining control of the system and using it against the network. |
| 1. Use Effective Quality Assurance Techniques | To validate a product or application, secure testing should be performed ahead of time to provide some resilience and identify weak points that can be approved. This can be accomplished via source code audits and penetration testing to ‘harden’ the code. Additionally, Independent reviewers should be utilized to provide for better coverage on the solution, and to provide an unbiased opinion. |
| 1. Adopt a Secure Coding Standard | A secure coding standard should be utilized when working on a solution. This ensures that everyone working on the project is building code to the same quality and is using proper precautions to ensure that a severe vulnerability is not introduced. |

## 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-CPP | Ensure that float conversions via cast are not accidental. |

| **Noncompliant Code** |
| --- |
| When using an implicit or explicit cast, we must guarantee the operation does not result in lost or incorrect data. Additionally, this code is non-compliant as we are unsure if this was requested cast (one that we want to perform. ) |
| void func(void) {  float num1 = 2147483647.123432;  int num2 = num1;  } |

| **Compliant Code** |
| --- |
| This code is complaint as it specifically denotes we are requesting a cast from int to a float with a loss off precision. |
| void func(void) {  float num1 = 2147483647.123432;  int num2 = static\_cast<int>(f);  } |

|  |
| --- |
| **Principles(s):** 1  This standard applies to the principal ‘**Validate Input Data**’ as we want to ensure that the conversion has proper input before trying to cast something that doesn’t make sense. This could cause buffer overflows that crash the program or cause undesirable behavior. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 1.66 | memsetValueOutOfRange | The second argument to memset() cannot be represented as unsigned char |
| Parasoft C/C++test | 2020.2 | CERT\_C-INT31-l | Avoid integer overflows |
| Coverity | 2017.07 | NEGATIVE\_RETURNS | Can find instances where an integer expression is implicitly converted to a narrower integer type, where the signedness of an integer value is implicitly converted, or where the type of a complex expression is implicitly converted |

### Coding Standard 2

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Data Value** | STD-002-CPP | For integer types (int, double, float, etc.), ensure division operations do not result in divide-by-zero. |

| **Noncompliant Code** |
| --- |
| This code is not compliant as it is a divide by zero situation and can end in an undefined result. |
| void func() {  int a = 5;  int b = 0;  int c = a / b;  } |

| **Compliant Code** |
| --- |
| This code is compliant as it makes sure that divide-by-zero errors are not able to cause undefined behavior. |
| void func() {  int a = 5;  int b = 0;  try  {  int c = a / b;  }  catch (runtime\_error &e)  {  // Error Handling  // Caught Divide-by-Zero  }  } |

|  |
| --- |
| **Principles(s):** **10**  The ‘**Develop a secure Coding Standard**’ applies here as its just bad practice to allow divide-by-zero errors to occur. This can end up with a number that is undefined and can cause other weird behavior if that number is used further down the line. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | int-division-by-zero  int-modulo-by-zero | Fully checked |
| Cppcheck | 1.66 | zerodiv  zerodivcond | Context sensitive analysis of division by zero  Not detected for division by struct member / array element / pointer data that is 0  Detected when there is unsafe division by variable before/after test if variable is zero |
| Parasoft C/C++test | 2020.2 | CERT\_C-INT33-a | Avoid division by zero |
| SonarQube C/C++ Plugin | 3.11 | S3518 |  |

### Coding Standard 3

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **String Correctness** | STD-003-CPP | Ensure all strings have a null terminator. |

| **Noncompliant Code** |
| --- |
| This code assumed that the buffer variable has a properly terminated but does not check for its existence and therefor non-compliant. |
| #include <iostream>    void f() {  **char** buf[12];    std::cin >> buf;  cout << buf;  } |

| **Compliant Code** |
| --- |
| This code is compliant as it uses strcpy to copy the buffer to the string variable temp. Using this type of copy always enforces the null terminator. |
| #include <iostream>    void f() {  std::string temp;  **char** buf[12];    std::cin >> buf;  std::strcpy( temp, source );  } |

|  |
| --- |
| **Principles(s):** 1, 10  This standard applies to the principal ‘**Validate Input Data**’ as we want to that strings that get passed into the program are correctly null terminated. If not null terminated, using functions that rely on the null terminator will fail or generate undefined behavior. ‘**Develop a Coding Standard**’ is also a valid principal, as generally speaking, we don’t want to have bad string data in any of the code we write, or is internal to our application. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Likely | Medium | High | 1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 2017.07 | STRING\_SIZE |  |
| Parasoft C/C++test | 2020.2 | CERT\_C-STR31-e | Avoid using unsafe string functions which may cause buffer overflows |
| PVS-Studio | 7.07 | V518, V645, V727, V755 |  |
| TrustInSoft Analyzer | 1.38 | mem\_access | Exhaustively verified (see one compliant and one non-compliant example). |

### Coding Standard 4

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **SQL Injection** | STD-004-CPP | Utilize parameters over raw string queries to access data sources. |

| **Noncompliant Code** |
| --- |
| This code is non-compliant as it injects raw strings into the query engine and can allow users to enter additional items in the string and cause it to fail. |
| #include <iostream>  #include <sqlite3.h>  void func()  {  sqlite3\* DB;     int exit = 0;     exit = sqlite3\_open("example.db", &DB);  string sql("INSERT INTO PERSON VALUES(1, 'STEVE', 'GATES', 30, 'PALO ALTO', 1000.0);"       exit = sqlite3\_exec(DB, sql.c\_str(), NULL, 0, &messaggeError);     if (exit != SQLITE\_OK) {         std::cerr << "Error Insert" << std::endl;         sqlite3\_free(messaggeError);     }  } |

| **Compliant Code** |
| --- |
| This code is compliant as it uses parameters instead of raw strings to insert data into the query. Its much harder to succumb to SQL injection style attacks. |
| #include <iostream>  #include <sqlite3.h>  void func()  {  sqlite3\* DB;     int exit = 0;     exit = sqlite3\_open("example.db", &DB);  string sql("INSERT INTO PERSON VALUES(1, 'STEVE', 'GATES', 30, 'PALO ALTO', 1000.0);"       exit = sqlite3\_exec(DB, sql.c\_str(), NULL, 0, &messaggeError);     if (exit != SQLITE\_OK) {         std::cerr << "Error Insert" << std::endl;         sqlite3\_free(messaggeError);     }  } |

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

|  |
| --- |
| **Principles(s):** 3, 7  The principal ‘**Architect and Design for Security Policies**’ applies to this standard as we are building this standard to prevent a specific type of attack vector called a SQL Injection attack. This is a great example of writing policies to counteract a specific type of attack. The ‘**Sanitize Data sent to other systems**’ also applies to this standard, as we do not want to send data (in this form a query) to another system that could compromise the data/system and return a potentially unexpected response. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Priority | Medium | 5 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Coverity | 7.5 | SQLI  FB.SQL\_PREPARED\_STATEMENT\_GENERATED\_  FB.SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Findbugs | 1.0 | SQL\_NONCONSTANT\_STRING\_PASSED\_TO\_EXECUTE | Implemented |
| Parasoft Jtest | 2020.2 | BD-SECURITY-TDSQL | Protect against SQL injection |

### Coding Standard 5

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Memory Protection** | STD-005-CPP | When using malloc/alloc, detect and handle memory allocation errors. |

| **Noncompliant Code** |
| --- |
| This code is noncompliant as we do not check to see if the allocation of memory actually succeed, and thus move on to attempt an operation. |
| #include <cstring>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = (int\*) malloc(5\*sizeof(int));  delete [] copy;  } |

| **Compliant Code** |
| --- |
| This code is compliant, as we check for the existence of a valid memory address inside the pointer, thus conforming to the standard. |
| #include <cstring>  #include <new>    void f(const int \*array, std::size\_t size) noexcept {  int \*copy = (int\*) malloc(5\*sizeof(int));  if (!copy) {  // Handle error  return;  }  delete [] copy;  } |

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

|  |
| --- |
| **Principles(s):** 5, 10  The principal of ‘**Heed Compiler Warnings**’ is important here because while in most cases accessing freed memory will result in an application crash, there are some cases where it won’t immediately. This can cause your application to access memory it should not have access to or overwrite values for other variables that are stored. The principal of ‘**Develop a Coding Standard**’ also applies here as this needs to be a general standard. Accessing freed memory will always end up being an issue. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Priority | Medium | Medium | 2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 1.66 | leakReturnValNotUsed | Doesn't use return value of memory allocation function |
| Parasoft C/C++test | 2020.2 | CERT\_C-MEM31-a | Ensure resources are freed |
| SonarQube C/C++ Plugin | 3.11 | S3584 |  |
| TrustInSoft Analyzer | 1.38 | malloc | Exhaustively verified. |

### Coding Standard 6

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Assertions** | STD-006-CPP | Do not leave assertions in production code |

| **Noncompliant Code** |
| --- |
| While assertions are normally turned off in production code, there have been exploits used during production builds to hijack and inject malicious code in the build. This is non compliant as the assert is in code flow and not in a test case. |
| void f()  {  std::vector<int> myVec\* = std::vector<int>(5);  assert(myVec != nullptr);  myVec.push\_back(1);  } |

| **Compliant Code** |
| --- |
| This code is compliant as it has the assert statement removed. |
| void f()  {  std::vector<int> myVec\* = std::vector<int>(5);  myVec.push\_back(1);  } |

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

|  |
| --- |
| **Principles(s):** 10  This is purely subjective based on personal experience when performing compilations, so I believe ‘**Developing a Coding Standard**’ is the only principal that applies in this scenario. We are limiting the exposure/issues that are code can potentially be exposed to, if these directives are applied in production. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Unlikely | Low | 1 | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 6.7 | S3346 | Expressions used in "assert" should not produce side effects |

### Coding Standard 7

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-007-CPP | Handle all exceptions |

| **Noncompliant Code** |
| --- |
| In this example, the target method nor the main function do not catch any exceptions thrown by raise\_exception(). Because there is no handler, the program terminates. |
| void raise\_exception() noexcept(false);    void target() {  raise\_exception();  }    int main() {  target();  } |

| **Compliant Code** |
| --- |
| In this example, the main method captures all exceptions using the unexpected error handler. Allows the program to terminate gracefully. |
| void raise\_exception() noexcept(false);    void target() {  raise\_exception();  }    int main() {  try {  target();  } catch (...) {  // Handle error  }  } |

|  |
| --- |
| **Principles(s):** 10  The principal of ‘**Develop a Coding Standard**’ applies to this standard as while not handling all exceptions is not detrimental to an application when everything works right, any unhandled or exceptions that occur to bad data will cause the application to crash and cause corruption. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Priority | Medium | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 | main-function-catch-all  early-catch-all | Partially checked |
| LDRA tool suite | 9.7.1 | 527 S | Partially implemented |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-ERR51-a  CERT\_CPP-ERR51-b | Always catch exceptions  Each exception explicitly thrown in the code shall have a handler of a compatible type in all call paths that could lead to that point |
| RuleChecker | 20.10 | main-function-catch-all  early-catch-all | Partially checked |

### Coding Standard 8

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Exceptions** | STD-008-CPP | Do not leak resources when handling exceptions. |

| **Noncompliant Code** |
| --- |
| This code is non-compliant due to the error handling not cleaning up the allocated myVector upon the operation throwing. |
| #include <vector>  void f() {  std::vector<int> \*myVector = new std::vector<int>(5);  try  {  myVector->at(1)  } catch (…) {  // Process error with logic  throw;  }  delete myVector;  } |

| **Compliant Code** |
| --- |
| This code is compliant with the standard, as the try/catch block destroys the allocated vector regardless of an error being thrown from at(1). This is accomplished via the use of a finally block attached to the try catch. |
| #include <vector>  void f() {  std::vector<int> \*myVector = new std::vector<int>(5);  try  {  myVector->at(1)  } catch (…) {  // Process error with logic  throw;  }  finally  {  // Handle the deletion of the allocated memory  delete myVector;  }  } |

|  |
| --- |
| **Principles(s):** 2, 9  The principals connected to this standard are ‘**Heed Compiler Warnings**’ and ‘**Use Effective Quality Assurance Techniques**’. Heeding compiler warnings in this instance will assist you in preventing this type of issue from occurring and prevent an outright crash in your application. Additionally, using effective testing in QA will show the issue and allow it to be properly fixed. |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Low | Priority | High | Low | 3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | ALLOC.LEAK | Leak |
| Helix QAC | 2021.1 |  |  |
| LDRA tool suite | 9.7.1 | 50 D | Partially implemented |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-ERR57-a | Ensure resources are freed |

### Coding Standard 9

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Iterators** | STD-009-CPP | Guarantee that container indices and iterators are within a valid range and are contiguous. |

| **Noncompliant Code** |
| --- |
| This code does not conform to the standard, as it is attempting to access an invalid index. |
| #include <vector>  void f() {  std::vector<int>\* myVector = new std::vector<int>(4);  int a = myVector->at(5);  delete myVector;  } |

| **Compliant Code** |
| --- |
| This code uses the correct 0-3 indices to select values stored in the vector. Thus, it conforms to the standard. |
| void f() {  std::vector<int>\* myVector = new std::vector<int>(4);  int a = myVector->at(1);  delete myVector;  } |

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

|  |
| --- |
| **Principles(s):** 10  I believe that this standard is related to the principal ‘**Adopt a clear Coding Standard**’as while programmatically its correct to index arrays/vectors in this fashion, you need to ensure you index within a valid range; specifically, when this index is dynamically computed. It makes sense to check the size before allowing the operation to continue, and to alert the user if the index is out-of-range. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| CodeSonar | 6.0p0 | LANG.MEM.BO  LANG.MEM.BU  LANG.MEM.TO  LANG.MEM.TU  LANG.MEM.TBA  LANG.STRUCT.PBB  LANG.STRUCT.PPE | Buffer overrun  Buffer underrun  Type overrun  Type underrun  Tainted buffer access  Pointer before beginning of object  Pointer past end of object |
| LDRA tool suite | 9.7.1 | 45 D, 47 S, 476 S, 489 S, 64 X, 66 X, 68 X, 69 X, 70 X, 71 X, 79 X | Partially implemented |
| Parasoft C/C++test | 2020.2 | CERT\_CPP-CTR50-a | Guarantee that container indices are within the valid range |
| PVS-Studio | 7.07 | V781 |  |

### Coding Standard 10

| **Coding Standard** | **Label** | **Name of Standard** |
| --- | --- | --- |
| **Input/Output** | STD-010-CPP | Do not access closed files. |

| **Noncompliant Code** |
| --- |
| Code is not compliant as the stdout stream is being used after its closed. |
| #include <stdio.h>    int close\_stdout(void) {  if (fclose(stdout) == EOF) {  return -1;  }    printf("stdout successfully closed.\n");  return 0;  } |

| **Compliant Code** |
| --- |
| Code is compliant as the stream is not being utilized after its closed. |
| #include <stdio.h>    int close\_stdout(void) {  if (fclose(stdout) == EOF) {  return -1;  }    fputs("stdout successfully closed.", stderr);  return 0;  } |

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

|  |
| --- |
| **Principles(s):** 10  I believe that this standard is related to the principal ‘**Adopt a clear Coding Standard**’as you should always check a resources status prior to using a resource. This should be standard practice to prevent undefined behaviors occurring from accessing the resources pointer. |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Astrée | 20.10 |  | Supported |
| Coverity | 2017.07 | USE\_AFTER\_FREE | Implemented |
| Polyspace Bug Finder | R2021a | CERT C: Rule FIO46-C | Checks for use of previously closed resource (rule partially covered) |
| SonarQube C/C++ Plugin | 3.11 | S3588 |  |

## Defense-in-Depth Illustration

This illustration provides a visual representation of the defense-in-depth best practice of layered security.



## 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.

Existing DevOps policies are a great starting point for implementing a DevSecOps platform. DevOps while great at releasing new software at the current operational speed does not do enough to address the security concerns of the modern company. Where DevSecOps improve on the DevOps formula is integrating security into the formula at the beginning. It causes an engineering culture shift and aims at weight security, software development and operations as equal partners. The main goal of DevSecOps is the application of security principals through all phases of Software Development Lifecycle, which you see as phases in the figure above.

To support the standards that we have added as part of this policy, I would make several modifications in each phase of our traditional DevOps lifecycle to bring it up to the standard of DevSecOps. For the phase of planning, I would brainstorm traditional security attack patterns (for example, SQL Injection or Man-in-the-Middle attacks) and come up with countermeasures to include in the product from its initial inception. For our develop and build steps, I would include our mitigation approaches from our coding standards and practice general secure coding practices to minimize instability and reduce how outside factors can affect our application via manipulation. For our test phase, I would embrace automated unit testing to handle the individual units of work (the smallest pieces of the application). Additionally, I would implement middle teir integration testing that tests the whole application stack from application to database, attempting to utilize popular strategies to gain access to the system. This would include, but not limited to SQL Injection and memory management attacks.

For our release, deploy, operate and monitor stages, I would utilize a secure container system to ensure no unauthorized access to the underlying OS, and use log collection and automated log sniffing to detect and stop instructions before they break the system. Additional network traffic can be utilized to determine if an abnormal amount of traffic hits a specific node and determine if a DDOS attack is being used to bring down the node or gain access.

## 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 | Low | Priority | High | Low | 2 |
| STD-002-CPP | Low | Likely | Medium | Low | 2 |
| STD-003-CPP | High | Likely | Medium | High | 1 |
| STD-004-CPP | High | Priority | Medium | 5 | 3 |
| STD-005-CPP | Medium | Priority | Medium | Medium | 2 |
| STD-006-CPP | Low | Unlikely | Low | 1 | 3 |
| STD-007-CPP | Low | Priority | Medium | Low | 3 |
| STD-008-CPP | Low | Priority | High | Low | 3 |
| STD-009-CPP | High | Likely | High | Medium | 2 |
| STD-010-CPP | Medium | Unlikely | Medium | Low | 3 |

## 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-at-rest** is a concept in where data that is not being actively utilized (say on a hard disk or database) is encrypted. The goal of this policy is to protect data in the event of a breech and these files are stolen. The attacker would have to brute force the encryption (which would take a long period of time) or gain access to the encryption keys to be able to decrypt the data and make it actionable. This type of process would take several years and limits the effectiveness of the data stolen. |
| Encryption at flight | **Encryption-in-flight** refers to the concept or protecting data as its moving around the network. For example, a web application pulling data from a database. This policy is important as it attempts to protect data while arguably at its most vulnerable point and subject to the most exposure. Protecting data in flight is achieved through the use of SSL/TLS connections between web server and database. Additional measure of protections is the use of VPN where network segments need to be joined. This ensures that normal packet sniffing applications like WireShark and TCPDump are not able to read the packets of the transmission and piece them together. |
| Encryption in use | **Encryption-in-Use** refers to the concept of protecting data when its actively being utilized. For example, when a web server has pulled data from a database and is making computations/delivering data to a consumer. Protecting data-in-use is accomplished through the use of programming techniques like utilizing protected memory (like .NET ProtectedMemory class) and Homomorphic encryption which allows manipulating encrypted strings as it were plaintext. Utilizing these safeguards limits attacking an application server in its usefulness, as you would need to breach these types of safeguards using brute force. This limits the data’s usefulness as this type of decryption can take years. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | **Authentication** is the process in which a server or application proves that you are who you say you are. Authentication is important as it determines who should and who should not have access to the application as a whole. This process is governed through a logon process. Usually, a user has a logon that is tied to their email address or specific user name. A password is provided to verify the login holder’s access. Additional forms of authentication should be included, such as 2 factor OTP or utilizing OAUTH technology for SSO. This policy should apply to all resources to prevent unauthorized access by a third party and to lock data behind a door where you need a key to access it. |
| Authorization | **Authorization** is the process that occurs after **Authentication** and grants the level of access that you have to a system, its files, and its resources. Authorization is used to control a user’s level of access, most often through role-based permissions, that define what a user can do. Tying this with the principal of **Default Deny** new users should not inherently have access to any resources even if authenticated. Only through the granting of a role should access be allowed to a particular resource. This policy is important to upheld as it prevents authenticated users without the proper authorization from accessing data, they should not be able to access. Additionally, it makes it very ease to ‘promote’ someone into a higher tier with better resource access. |
| Accounting | **Accounting** is the process of tracking changes to a particular system or resource. An example of accounting is tracking what files are accessed by users and what changes are made to database files. While systems can be built to provide this functionality, there are many software-based applications that provide aggregation for this data and allow reports to be run off of it. An example of this type of system is Thycotic Secret Server. This is important system to have as its both informative before and after a breach. If a specific user account is accessing a resource before a breech, having this information allows the security team to scrutinize this access and potentially prevent a breech. Immediately after a breech, this information is important to performing an root cause analysis on the breach, and to seal that pathway up as fast as possible. This policy applies to our scenario, as we want to protect unauthorized intrusion on our protected data. |

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

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

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

# Audit Controls and Management

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

Evidence will include the following:

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

# Enforcement

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

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

# Exceptions Process

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

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

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

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

# Distribution

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

# Policy Change Control

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

# Policy Version History

| Version | Date | Description | Edited By | Approved By |
| --- | --- | --- | --- | --- |
| 1.0 | 08/05/2020 | Initial Template | David Buksbaum |  |
| 1.1 | 04/17/2021 | Initial Milestone Completion | Michael Mihalik | David Buksbaum |
| 1.2 | 04/24/2021 | Security Policy Completion | Mihcael Mihalik | David Buksbaum |

# Appendix A Lookups

## Approved C/C++ Language Acronyms

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