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

**Joshua Perez | SNHU CS-405-H7071**



# Green Pace Secure Development Policy

## Contents

[Overview 2](#_Toc52464053)

[Purpose 2](#_Toc52464054)

[Scope 2](#_Toc52464055)

[Module Three Milestone 2](#_Toc52464056)

[Ten Core Security Principles 2](#_Toc52464057)

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

[Coding Standard 1 4](#_Toc52464059)

[Coding Standard 2 5](#_Toc52464060)

[Coding Standard 3 6](#_Toc52464061)

[Coding Standard 4 7](#_Toc52464062)

[Coding Standard 5 8](#_Toc52464063)

[Coding Standard 6 9](#_Toc52464064)

[Coding Standard 7 10](#_Toc52464065)

[Coding Standard 8 11](#_Toc52464066)

[Coding Standard 9 13](#_Toc52464067)

[Coding Standard 10 14](#_Toc52464068)

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

[Project One 15](#_Toc52464070)

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

[2. Risk Assessment 15](#_Toc52464072)

[3. Automated Detection 15](#_Toc52464073)

[4. Automation 15](#_Toc52464074)

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

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

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

[Audit Controls and Management 18](#_Toc52464078)

[Enforcement 18](#_Toc52464079)

[Exceptions Process 18](#_Toc52464080)

[Distribution 19](#_Toc52464081)

[Policy Change Control 19](#_Toc52464082)

[Policy Version History 19](#_Toc52464083)

[Appendix A Lookups 19](#_Toc52464084)

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

## Overview

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

## Purpose

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

## Scope

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

## Module Three Milestone

### Ten Core Security Principles

| **Principles** | Write a short paragraph explaining each of the 10 principles of security. |
| --- | --- |
| 1. ValidateInput Data | This principle emphasizes the importance of treating all input data as potentially malicious. It involves rigorously checking, sanitizing, and validating data received from external sources before processing. This practice helps prevent attacks like SQL injection, cross-site scripting, and buffer overflows, which exploit unvalidated inputs. |
| 1. Heed Compiler Warnings | Compiler warnings often highlight potential security vulnerabilities or questionable practices in code. By paying attention to and addressing these warnings, developers can proactively rectify issues that might lead to security flaws, ensuring more robust and secure software. |
| 1. Architect and Design for Security Policies | This principle focuses on incorporating security considerations into the software architecture and design phases. It involves designing systems with security policies in mind, like ensuring data encryption, secure communication protocols, and proper authentication methods, to create a strong foundation for secure software. |
| 1. Keep It Simple | Complexity often leads to security vulnerabilities. This principle advocates for simplicity in code and system design to make it easier to understand, maintain, and secure. Simplifying systems reduces the chances of errors that could be exploited by attackers. |
| 1. Default Deny | In access control, the 'default deny' principle means that system access should be denied by default and granted only when explicitly allowed. This conservative approach minimizes the risk of unauthorized access, ensuring that only permitted entities have access to resources. |
| 1. Adhere to the Principle of Least Privilege | This principle dictates that users and programs should be granted only the minimum privileges necessary to perform their tasks. Limiting privileges reduces the risk and impact of security breaches by restricting access to critical system components. |
| 1. Sanitize Data Sent to Other Systems | Ensuring the cleanliness of data sent to other systems or components is vital. This involves removing or encoding potentially harmful data elements to prevent security vulnerabilities, like injection attacks, in the receiving system. |
| 1. Practice Defense in Depth | Defense in depth is a layered approach to security. It involves implementing multiple, overlapping security measures to protect the integrity of the information. If one layer fails, others still stand, providing a comprehensive security strategy. |
| 1. Use Effective Quality Assurance Techniques | This principle underscores the importance of rigorous quality assurance processes in software development. It involves thorough testing, code reviews, and analysis to identify and rectify security vulnerabilities before deployment. |
| 1. Adopt a Secure Coding Standard | Following a secure coding standard helps in preventing common security flaws. These standards provide guidelines and best practices for writing secure code and are crucial in developing robust, secure applications resistant to common exploits. |

### 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** | **Strong Typing Enforcement** |
| --- | --- | --- |
| **Data Type** | STD-001-DAT | Using strong typing ensures that each variable is declared with a specific type, reducing errors and vulnerabilities related to type mismatches or conversions. |

| **Noncompliant Code** |
| --- |
| Variables are loosely typed, leading to potential type conversion errors or vulnerabilities. |
| auto userAge = "18"; // Implicitly treating a string as an integer |

| **Compliant Code** |
| --- |
| Explicitly declaring variable types to prevent unintended type conversions. |
| int userAge = 18; // Correctly declaring age as an integer |

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

| **Principles(s):** DAT-STD-001: Enforces strong typing to prevent errors and vulnerabilities due to type mismatches or conversions. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | Strong Typing Rules | SonarQube can be configured to identify and flag instances in the code where implicit type conversions could occur, ensuring that variables are explicitly typed according to the strong typing standard. |
| ESLint | 8.0 | Type Checking with TypeScript | When used in conjunction with TypeScript, ESLint can check for type-related issues in JavaScript code, enforcing strict type discipline as per the project's coding standards. |
| ReSharper for C# | 2022.1 | Type Usage Analysis | ReSharper provides extensive code inspection capabilities that include enforcing strong typing disciplines, thus aligning C# code with the specified coding standards. |

#### Coding Standard 2

| **Coding Standard** | **Label** | **Validate Data Values** |
| --- | --- | --- |
| **Data Value** | STD-002-DAT | Ensuring data values conform to expected ranges or formats can prevent issues like buffer overflows or incorrect data processing. |

| **Noncompliant Code** |
| --- |
| Data value is not validated for the expected range. |
| int userAge = getUserAge(); // No range check |

| **Compliant Code** |
| --- |
| Adding range checks to validate data values. |
| int userAge = getUserAge();  if (userAge < 0 || userAge > 130) {  throw std::range\_error("Invalid age");  } |

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

| **Principles(s):** STD-002-DAT: Validates that data conforms to defined ranges to prevent errors like buffer overflows and incorrect data processing. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ESLint with custom rules | 8.0 | Custom Range Validation | ESLint can be extended with custom rules to ensure range validations are implemented. |
| SonarQube | 9.5 | Data Range Validation | SonarQube's static analysis can detect when numerical values are outside of a defined safe range. |
| CodeQL | 2.7.4 | Numeric Value Range Analysis | CodeQL allows the creation of custom queries to detect issues with data ranges across multiple languages. |

#### Coding Standard 3

| **Coding Standard** | **Label** | **Safe String Handling** |
| --- | --- | --- |
| **String Correctness** | STD-003-STR | Proper string handling prevents vulnerabilities like buffer overflows and ensures data integrity. |

| **Noncompliant Code** |
| --- |
| Unsafe string operations that can lead to buffer overflows. |
| char username[10];  strcpy(username, userInput); // Risk of buffer overflow |

| **Compliant Code** |
| --- |
| Using safe string functions to prevent buffer overflows. |
| char username[10];  strncpy(username, userInput, sizeof(username) - 1);  username[sizeof(username) - 1] = '\0'; // Ensuring null termination |

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

| **Principles(s):** STD-003-STR: Enforces the use of safe string operations to prevent common vulnerabilities such as buffer overflows. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | P3 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Flawfinder | 2.0.11 | String Vulnerabilities | Scans C/C++ code for common insecure string handling patterns. |
| Coverity | 2021.12 | Tainted Data Analysis | Identifies unsafe string manipulations that could lead to security vulnerabilities like buffer overflows. |
| Flawfinder | 2.0.11 | Unsafe String Functions Check | Flawfinder scans C/C++ source code and identifies potential security flaws, specifically targeting unsafe string manipulation functions. |

#### Coding Standard 4

| **Coding Standard** | **Label** | **SQL Injection Prevention** |
| --- | --- | --- |
| **SQL Injection** | STD-004-SQL | Preventing SQL injection protects the database from unauthorized access or modification. |

| **Noncompliant Code** |
| --- |
| Directly using user input in SQL queries can lead to SQL injection. |
| std::string query = "SELECT \* FROM users WHERE username = '" + userInput + "'"; |

| **Compliant Code** |
| --- |
| Using prepared statements to safely handle SQL queries. |
| std::string query = "SELECT \* FROM users WHERE username = ?";  // Use prepared statements with query |

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

| **Principles(s):** STD-004-SQL: Prevents SQL injection by sanitizing user input and using prepared statements. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | High | P1 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| SonarQube | 9.5 | SQL Injection Rules | SonarQube provides static code analysis to detect vulnerabilities including SQL injection risks, enforcing secure coding standards. |
| Static Code Analyzer | 21.1.0 | SQL Injection Rules | Analyzes source code to identify SQL injection vulnerabilities, providing a detailed vulnerability report. |
| Checkmarx | 9.4.2 | SQL Injection Detection | Utilizes static application security testing to pinpoint SQL injection flaws and other security vulnerabilities. |

#### Coding Standard 5

| **Coding Standard** | **Label** | **Memory Allocation and Deallocation** |
| --- | --- | --- |
| **Memory Protection** | STD-005-MEM | Proper management of memory allocation and deallocation prevents leaks and corruption. |

| **Noncompliant Code** |
| --- |
| Potential memory leak due to not freeing allocated memory. |
| char\* buffer = new char[1024];  // Missing delete for buffer |

| **Compliant Code** |
| --- |
| Correctly managing memory allocation and deallocation. |
| char\* buffer = new char[1024];  // Operations on buffer  delete[] buffer; |

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

| **Principles(s):** STD-005-MEM: Ensures dynamic memory is properly managed to avoid leaks and corruption. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | P5 | L3 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Valgrind | 3.18.1 | Memcheck | An instrumentation framework for building dynamic analysis tools that can detect memory mismanagement. |
| AddressSanitizer (part of LLVM/Clang) | 14.0.0 | AddressSanitizer | A fast memory error detector that can detect memory leaks. |
| Visual Studio | 2022 | Code Analysis Memory Checks | Integrated tool in Visual Studio that can analyze code for memory allocation issues. |

#### Coding Standard 6

| **Coding Standard** | **Label** | **Use of Assertions** |
| --- | --- | --- |
| **Assertions** | STD-006-ASS | Assertions help catch unexpected conditions at runtime, aiding in debugging and robustness. |

| **Noncompliant Code** |
| --- |
| Omitting assertions where assumptions are made about program state. |
| int divide(int a, int b) {  return a / b; // No assertion for b != 0  } |

| **Compliant Code** |
| --- |
| Using assertions to validate assumptions. |
| int divide(int a, int b) {  assert(b != 0 && "Divisor should not be zero");  return a / b;  } |

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

| **Principles(s):** STD-006-ASS: Uses assertions to validate assumptions within the code to prevent incorrect program behavior. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| Medium | Low | Low | P4 | L4 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Cppcheck | 2.7 | Assert Usage Checks | Static analysis tool for C/C++ that checks for conditions that should be asserted in the code. |
| Clang-Tidy | 14.0.0 | bugprone-assert-side-effect | Checks that assertions are not producing side effects and are used correctly. |

#### Coding Standard 7

| **Coding Standard** | **Label** | **Proper Exception Handling** |
| --- | --- | --- |
| **Exceptions** | STD-007-EXC | Properly handling exceptions ensures the program can gracefully deal with runtime errors. |

| **Noncompliant Code** |
| --- |
| Not handling potential exceptions can lead to unexpected program termination. |
| int value = std::stoi(userInput); // May throw an exception if input is not a valid integer |

| **Compliant Code** |
| --- |
| Implementing exception handling to manage runtime errors. |
| int value;  try {  value = std::stoi(userInput);  } catch (const std::invalid\_argument& e) {  // Handle the exception  } |

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

| **Principles(s):** STD-007-EXC: Ensures that exceptions are handled properly to maintain program stability. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | P3 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| ReSharper C++ | 2022.1 | Exception Handling Analysis | ReSharper C++ is an extension for Microsoft Visual Studio that provides on-the-fly code analysis for C++, including exception handling. |
| CodeSonar | 6.2 | Exception Usage | Static analysis tool that identifies potential exceptions that are not properly handled. |
| SonarQube | 9.5 | Exception Handling Rules | SonarQube can be configured to ensure that all exceptions are caught and handled properly. |

#### Coding Standard 8

| **Coding Standard** | **Label** | **Maintain Adequate Code Documentation** |
| --- | --- | --- |
| Code Documentation | STD-008-COD | Proper documentation within code improves readability, maintainability, and helps in identifying potential security flaws. |

| **Noncompliant Code** |
| --- |
| Lack of comments and documentation in complex or critical code sections. |
| void processData(int data) {  // Complex logic with no comments  } |

| **Compliant Code** |
| --- |
| Providing clear comments and documentation for complex code sections. |
| void processData(int data) {  // Explanation of the processing logic  // and any relevant security considerations  } |

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

| **Principles(s):** STD-008-COD: Proper code documentation is crucial for understanding the purpose and functionality of code segments, particularly for complex logic that may have security implications. |
| --- |

**Threat Level**

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

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Doxygen | 1.9.4 | Documentation Coverage | Generates documentation from annotated C++ sources and can report on undocumented members. |
| Javadoc (for Java) | 17 | DocLint | Provides an API documentation generator and also includes a doc comment checking tool. |
| ESLint (for JavaScript) | 8.0 | valid-jsdoc | Lints JS code to ensure that comments are complete and properly formatted. |

**Coding Standard 9**

| **Coding Standard** | **Label** | **Safe File Handling** |
| --- | --- | --- |
| Secure File Operations | STD-009-FIL | Ensuring secure practices in file operations prevents vulnerabilities like path traversal attacks and unauthorized access. |

| **Noncompliant Code** |
| --- |
| Unsafe handling of file paths which could lead to security vulnerabilities. |
| std::ifstream file(userInput); // No validation of user input for file path |

| **Compliant Code** |
| --- |
| Validating file paths and handling files securely. |
| // Validate userInput for safe file path  std::ifstream file(sanitizedInput); |

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

| **Principles(s):** STD-009-FIL: Ensures file operations are performed securely, guarding against vulnerabilities like path traversal and unauthorized file access. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | Medium | Medium | P2 | L2 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Fortify Static Code Analyzer | 21.1.0 | Path Manipulation | Identifies potential security issues in file handling, such as path traversal. |
| Veracode | 10.0 | Insecure File Handling | Analyzes binary and bytecode to find security flaws in file operations. |

#### Coding Standard 10

| **Coding Standard** | **Label** | **Secure Network Practices** |
| --- | --- | --- |
| Secure Network Communication | STD-010-NET | Implementing secure network communication methods to prevent data breaches and protect sensitive information. |

| **Noncompliant Code** |
| --- |
| Using unencrypted, insecure network protocols for transmitting sensitive data. |
| // Using HTTP for sensitive data transmission  sendDataOverHTTP(sensitiveData); |

| **Compliant Code** |
| --- |
| Employing encrypted, secure protocols for network communication. |
| // Using HTTPS for secure data transmission  sendDataOverHTTPS(sensitiveData); |

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

| **Principles(s):** STD-010-NET: Secure network communication practices are enforced to prevent data breaches and protect sensitive information during transit. |
| --- |

**Threat Level**

| **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- |
| High | High | High | P1 | L1 |

**Automation**

| **Tool** | **Version** | **Checker** | **Description Tool** |
| --- | --- | --- | --- |
| Wireshark | 3.6.5 | Protocol Analyzer | Analyzes network protocols and can detect unencrypted data transmissions. |
| Nessus | 10.0 | Encrypted Communications Audit | Scans for unencrypted data transmission vulnerabilities and checks for proper encryption use. |
| Qualys | 10.0 | SSL/TLS Security Check | Provides a thorough assessment of the SSL/TLS configuration of a web server. |

### 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.

To enhance Green Pace's existing DevOps processes, automation tools can be incorporated at strategic points to ensure continuous security. In the Assess and Plan stage, automated tools can be integrated to perform static code analysis, ensuring that new code commits adhere to the security policy from the outset. This can be achieved by using tools like SonarQube, which can scan the source code for potential security issues, code smells, and bugs. During the Design phase, automated design analysis tools can validate the security posture of the architecture, ensuring that it adheres to the defined standards. Automated threat modeling tools can also be employed to predict potential threats and suggest mitigations. In the Build phase, Continuous Integration (CI) pipelines can be configured to include security checks as part of the build process. SAST tools should be used here to analyze the source code against the defined standards. Any issues detected at this stage can be automatically reported back to developers for remediation. The Verify and Test stage should leverage Dynamic Application Security Testing (DAST) tools to automatically test the running application for vulnerabilities. Prior to Pre-production, security configurations can be automatically applied and validated, and penetration testing tools can be scheduled to run regularly. In the Transition and Health Check phase, automated scripts should check that the security standards are implemented correctly in the staging environment. In Production, automated monitoring tools should be used to detect deviations from the standards in real time. Finally, during the Maintain and Stabilize phase, tools should be used to automatically compare the current state of the system with the defined security baseline, ensuring compliance. Automated incident response tools should be in place to handle any detected issues, enforcing standards by reverting to a known good state if necessary.

### 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-002-DAT | High | Low | Low | P4 | L3 |
| STD-003-STR | High | Medium | Medium | P3 | L1 |
| STD-004-SQL | High | High | High | P1 | L1 |
| STD-005-MEM | High | Medium | Medium | P5 | L3 |
| STD-006-ASS | Medium | Low | Low | P4 | L4 |
| STD-007-EXC | High | Medium | Medium | P3 | L2 |
| STD-008-COD | Medium | Medium | Low | P4 | L3 |
| STD-009-FIL | High | Medium | Medium | P2 | L2 |
| STD-010-NET | High | High | High | P1 | L1 |

### 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 critical security measure which ensures that all sensitive data, when stored on any device or media, is not readily accessible or decipherable to unauthorized users. This policy mandates that any such stored data, whether it be on internal servers, cloud storage, external backup media, or any other form of digital storage, must be encrypted using industry-standard algorithms and secure key management practices. The reason for enforcing this policy is to protect against data breaches that can occur if storage devices are lost, stolen, or otherwise compromised. Compliance with this policy not only safeguards our data against unauthorized access but also aligns with regulatory requirements for data protection and privacy. |
| Encryption at flight | Encryption in transit, often referred to as encryption in flight, is the practice of encrypting data while it is being transmitted over a network. The policy here requires that any data sent over the network, both within our internal networks and across the internet, must be encrypted. This is to protect the confidentiality and integrity of data as it moves from one location to another, preventing potential interception by unauthorized parties. To achieve this, we employ secure communication protocols like HTTPS, SSH, and TLS for all data transmissions. The encryption in flight policy is crucial for maintaining the trust of our clients and partners, ensuring that all data exchanges meet the high-security standards expected in today’s digital landscape. |
| Encryption in use | Encryption in use pertains to protecting data that is currently being processed. This policy mandates that data, especially of a sensitive nature, must be encrypted not just when at rest or in transit, but also when in memory and during computation. This is to prevent data exposure in the event of system compromises such as memory scrapes or if an unauthorized entity gains access to a system’s computational resources. We enforce this policy by using advanced encryption methods and ensuring that any temporary data is securely handled. Encrypting data in use is a forward-thinking approach to security, addressing emerging threats and reducing the attack surface within active computing environments. |

| 1. **Triple-A Framework\*** | **Explain what it is and how and why the policy applies.** |
| --- | --- |
| Authentication | Authentication is the mechanism by which a system verifies the identity of a user trying to gain access. Our policy requires users to authenticate themselves using robust methods, such as multi-factor authentication, to ensure that system access is granted only to legitimate users. By logging and monitoring all authentication attempts, we maintain a secure and verifiable access control system. This policy is pivotal to our security posture, as it prevents unauthorized access by ensuring that only verified individuals can enter our systems, thus protecting sensitive information from potential malicious actors. |
| Authorization | The authorization policy in place ensures that users are granted access only to the resources necessary for their specific role or task, adhering to the principle of least privilege. This policy applies each time a user attempts to interact with resources, whether accessing databases, modifying files, or altering configurations. By enforcing role-based access controls and keeping meticulous records of permissions and their changes, we minimize the risk of unauthorized or inappropriate access to sensitive resources. Authorization is a key component of our overall security strategy, as it defines and restricts user actions based on their authenticated identity, thereby enhancing our system's integrity and data protection capabilities. |
| Accounting | Accounting, in the context of IT security, refers to the tracking and logging of user activities within our systems, specifically focusing on interactions that could affect the security posture, such as database modifications or file access. The policy for accounting is to maintain detailed and tamper-evident logs for all significant actions, which enables us to audit user behavior, track changes, and detect potential security incidents. The application of this policy is vital for accountability, as it allows us to perform thorough security audits, comply with regulatory standards, and swiftly respond to any anomalies or breaches. Keeping rigorous logs is not only a best practice for security but also crucial for post-event analysis and forensics. |

**\***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
* **(Data Type) STD-001-DAT**:
  + Principle(s): 1 (DAT-STD-001), 9 (STD-009-FIL), 10 (STD-010-NET)
  + Justification: This 1st principle directly maps to the standard, emphasizing the importance of strong typing in programming to avoid type-related errors and security vulnerabilities. By enforcing strong typing, the standard minimizes the risks associated with type coercion and conversion mistakes. Principles 9 and 10 emphasize secure handling and transfer of data which are cross-cutting concerns relevant to Data Type and Data Value since securely handling and transmitting data is paramount regardless of the data's state.
* **(Data Value) STD-002-DAT**:
  + Principle(s): 2 (STD-002-DAT), 9 (STD-009-FIL), 10 (STD-010-NET)
  + Justification: The 2nd principle, that data must conform to expected ranges aligns with this standard's focus on preventing common programming errors like buffer overflows, which are often caused by unexpected data values. It reinforces the need for validation checks within the code. Principles 9 and 10 apply for the reasons mentioned above.
* **(String Correctness) STD-003-STR**:
  + Principle(s): 3 (STD-003-STR)
  + Justification: This principle directly corresponds to the standard by requiring the use of safe string operations, such as bounds-checked functions, to prevent security issues such as buffer overflows, thereby ensuring the integrity and security of string data.
* **(SQL Injection) STD-004-SQL**:
  + Principle(s): 4 (STD-004-SQL)
  + Justification: The principle of sanitizing user input to prevent SQL injection is the foundation of this standard. It supports the standard by dictating the use of secure coding practices, such as prepared statements, to protect against SQL injection vulnerabilities.
* **(Memory Protection) STD-005-MEM**:
  + Principle(s): 5 (STD-005-MEM), 7 (STD-007-EXC)
  + Justification: This 5th principle's focus on the proper management of dynamic memory supports the standard by aiming to prevent memory leaks and corruption. It highlights the importance of memory allocation and deallocation practices to maintain application stability. While focused on exception handling, the 7th principle is also relevant to Memory Protection, as proper exception handling can prevent memory corruption by handling errors gracefully instead of crashing a program.
* **(Assertions) STD-006-ASS**:
  + Principle(s): 6 (STD-006-ASS), 1 (DAT-STD-001), 3 (STD-003-STR), 4 (STD-004-SQL), 5 (STD-005-MEM)
  + Justification: The use of assertions to validate code assumptions directly supports this standard by ensuring that the code behaves as expected during runtime and aids in catching conditions that could lead to bugs or unexpected behavior. Principle 1 can also apply to standards such as Assertions since strong typing is crucial in accurately capturing and handling errors or exceptions within the code. Principles 3 and 4, while specific to string operations and SQL queries, also carry a broader message of input validation that is essential for preventing a range of vulnerabilities. They could support standards like Assertions, which could assert the validity of string lengths or query formats. Principle 5 applies because proper memory management is a concern that can affect various areas.
* **(Exceptions) STD-007-EXC**:
  + Principle(s): 7 (STD-007-EXC), 1 (DAT-STD-001)
  + Justification: Proper exception handling is crucial for the resilience of programs, and this principle underpins the standard by detailing the need for structured error handling mechanisms that allow a program to continue operation or fail gracefully. Principle 1 can also apply to standards such as Assertions since strong typing is crucial in accurately capturing and handling errors or exceptions within the code.
* **(Code Documentation) STD-008-COD**:
  + Principle(s): 8 (STD-008-COD)
  + Justification: Clear and thorough code documentation as dictated by this principle aligns with the standard by improving code maintainability and understanding, which is especially important for complex logic that may have security implications.
* **(Secure File Operations) STD-009-FIL**:
  + Principle(s): 9 (STD-009-FIL), 2 (STD-002-DAT)
  + Justification: The principle ensures secure file handling practices, which supports the standard's objective to guard against file-related vulnerabilities, such as path traversal and unauthorized access, through secure file operation procedures. Beyond validating data values, principle 2 can extend to Secure File Operations, as ensuring data conforms to expected formats or ranges is also vital when receiving file inputs.
* **(Secure Network Communication) STD-010-NET**:
  + Principle(s): 10 (STD-010-NET), 2 (STD-002-DAT)
  + Justification: This principle addresses the need for secure network communication to prevent data breaches, supporting the standard by mandating the use of secure protocols and methods to protect data as it traverses the network. Similar to what I stated in standard 9, principle 2 can extend to Secure Network Communication because ensuring data conforms to expected formats or ranges is also vital when receiving data over the network.

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 |  |
| 1.5 | 11/19/2023 | Module 3 | Joshua Perez |  |
| 2.0 | 12/8/2023 | Project 1 | Joshua Perez |  |

## Appendix A Lookups

### Approved C/C++ Language Acronyms

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