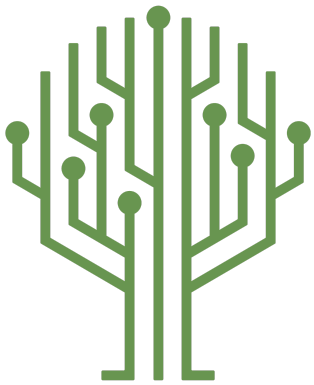


Green Pace

Green Pace Secure Development Policy

Contents

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.

Scope

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

Project One

Ten Core Security Principles

Principle	Description
1. Validate Input Data	Validating input data involves several steps. First, the system must collect input data from the user or administrator. Then, the system analyzes that information based on parameters set by the system administrator. After analysis, the system should decide on the method of data handling. If the data meets system requirements, the system should proceed with regular operations. If not, the system should return an error message to the user with an explanation. For example, a system should be able to validate input data for a field asking for a phone number, rejecting any non-numerical input.
2. Heed Compiler Warnings	When programming in the integrated development environment (IDE), compiler warnings may appear next to lines of code based on syntax or reference errors. Developers should pay attention to these errors since they usually catch the earliest instance of a system error. One example of a compiler warning includes an unfound variable, which the developer may not have initialized, or the developer misspelled the intended variable.
3. Architect and Design for Security Policies	Architecting and designing for security policies provides the blueprint for cybersecurity in a system. This provides the layers or elements of security a system may need for proper function and preservation of data. For example, one security design could be different kinds of user permissions based on the user's position in the system: administrator, supervisor, employee, etc.
4. Keep It Simple	To keep a system running efficiently and securely, simplest is best. Overcomplicating cybersecurity operations may prevent users from utilizing or accessing the system in a timely manner. For example, requiring a user to log into the system and at critical checkpoints is much simpler and efficient than requiring the user to input their credentials for every access point.
5. Default Deny	Default deny means that networks should block any activity or traffic that has not been approved by the system. This may come in the form of firewalls, which block all traffic except those which the system administrator or cybersecurity professional approved. Common examples of default deny are library and workplace firewalls on system devices.
6. Adhere to the Principle of Least Privilege	This assumes that all users in a system are given the least user privileges by default. This ensures that all users will not be granted excessive or full access upon initial account registration. User permissions can always be granted or demoted after the fact. For example, all users of the SNHU system may only be given "student" level privileges. Then, depending on a higher role, the user may have a different level of access than the other higher roles, such as instructor, advisor, dean, and director.
7. Sanitize Data Sent to Other Systems	Sanitization of data means that data is purposely deleted or destroyed so that it cannot be recovered. This ensures that confidential information is not stored somewhere it should not be. If it remains, infiltrators can recover and retrieve the data for malicious purposes. One example is editing a password in the login process – simply backspacing to the last point where the sequence was correct allows the user to proceed with the remaining valid characters.

Principle	Description
8. Practice Defense in Depth	Defense in depth involves several, independent layers of security in a system. This ensures protection of the system and its data, as well as the integrity of the data. If one layer fails, the other layers will be robust enough to continue protecting the system. For example, if a denial-of-service attack occurs, a system may be designed so that while the login process may be compromised, the system's data is still protected by other layers.
9. Use Effective Quality Assurance Techniques	Effective quality assurance techniques allow a system to be checked for data handling or versions of their dependencies. This includes routine maintenance and monitoring of data and the dependencies. When data is checked for quality assurance, this helps the system retain the right information. When dependencies are checked for quality assurance, this ensures that the system is using the latest versions of the dependency.
10. Adopt a Secure Coding Standard	Using the best programming practices when developing a system helps administrators and users ensure that the system is secure and efficient. Some standards include those previously mentioned above, such as data validation, default deny, data sanitization, and defense in depth. More than one standard can be used simultaneously and is often recommended to ensure an exponentially secure system in the long run.

C/C++ Ten Coding Standards

Coding Standard 1: Data Type

Coding Standard	Label	Name of Standard
Data Type	[STD-001-CPP]	This ensures that input and output data are safely collected, stored, and maintained based on the kind of data it is.

Noncompliant Code

Variables are initialized based on the correct or most logical data type.

```
string userFirstName
string userLastName
string userAddress
int userAccountNumber
int userRoutingNumber
```

Compliant Code

Variables are not initialized correctly or logically.

```
int userFirstName
int userLastName
int userAddress
string userAccountNumber
string userRoutingNumber
```

Principle 4: Keep It Simple
Data types should be logically assigned to respective variables. This helps the system minimize data corruption if values are cross-written, overwritten, or handled simultaneously.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	Medium	2	3

Automation

Tool	Version	Checker	Description Tool
Astree	23.04	type-compatibility type-compatibility-link distinct-extern	Fully checked

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	7.2.0	CertC-DCL40	Fully implemented
CodeSonar	7.4p0	LANG.STRUCT.DECL.IF LANG.STRUCT.DECL.IO	Inconsistent function declarations Inconsistent object declarations
Coverity	2017.07	MISRA C 2012 Rule 8.4	Implemented

Coding Standard 2: Data Value

Coding Standard	Label	Name of Standard
Data Value	[STD-002-CPP]	Data value standards allow the system to handle and analyze data more efficiently. It allows for accuracy of data during intake and output.

Noncompliant Code

Data value has an established length at the point of intake. Allows the system to obtain the data based on the specified value(s).

```
const std::string account_number = "CharlieBrown42";
char user_input[21]; // Change user input length to 21 characters
std::cout << "Enter a value: ";
std::cin.getline(user_input, 21);  // Only reads first 20 characters
                                   of input. Prevents data compromise
                                   to account_number.
```

Compliant Code

Data value does not have an established length. Allows the system to obtain data of any specification or length. This can result in a denial-of-service attack if misused.

```
const std::string account_number = "CharlieBrown42";
std::cout << "Enter a value: ";
std::cin >> user_input;  // Takes any length of account_number
```

Principle 9: Use Effective Quality Assurance Techniques
Allows the system to be checked for proper data input. Prevents data overflow which could result in common issues like denial of service attacks.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	18	1

Automation

Tool	Version	Checker	Description Tool
Astree	23.04		Supported Reports all buffer overflows resulting from copying data to a buffer not large enough to hold that data.

Tool	Version	Checker	Description Tool
Axivion Bauhaus Suite	7.2.0	CertC-STR31	Detects calls to unsafe string functions causing buffer overflow. Detects possible buffer overruns including those caused by unsafe usage of fscanf().
CodeSonar	7.4p0	LANG.MEM.BO LANG.MEM.TO MISC.MEM.NTERM BADFUNC.BO.*	Buffer overrun Type overrun No space for null terminator Collection of warning classes reporting uses of library functions prone to internal buffer overflows
Compass/ROSE			Can detect violations of data values but cannot handle cases involving strcpy_s() or manual string copies

Coding Standard 3: String Correctness

Coding Standard	Label	Name of Standard
String Correctness	[STD-003-CPP]	Allows string variables to be analyzed for the system to select the proper decision branching choice(s). Also validates data when needed.

Noncompliant Code

Program properly checks for string correctness and proceeds with secure method.

```
if (accountType == accountType) {
     displayAccountMenu();
}
else {
     cout << "Error: Account type could not be validated.";
     displayMainMenu();
}
```

Compliant Code

Program does not check for string correctness based on language standards and does not allow system to proceed properly.

```
if (accountType = accountType) {
     displayAccountMenu();
}
else {
     cout << "Error: Account type could not be validated.";
     displayMainMenu();
}
```

Principle 1: Validate Input Data
Ensures strings are analyzed and handled without possible data corruption.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Likely	Low	9	2

Automation

Tool	Version	Checker	Description Tool
Astree	23.04	string-literal-modification write-to-string-literal	Fully checked
Axivion Bauhaus Suite	7.2.0	CertC-STR30	Fully implemented
Compass/ROSE			Can detect simple violations of string correctness
Coverity	2017.07	PW	Deprecates conversion from a string literal to “char”

Coding Standard 4: SQL Injection

Coding Standard	Label	Name of Standard
SQL Injection	[STD-004-CPP]	Allows the system to be checked for SQL Injection vulnerabilities throughout any point. Protects the system and its data from compromise.

Noncompliant Code

The system contains a method for monitoring any vulnerabilities where an SQL Injection may occur.

```
int main()
{
    // initialize random seed:
    srand(time(nullptr));

    int return_code = 0;
    std::cout << "SQL Injection Example" << std::endl;

    // the database handle
    sqlite3* db = NULL;
    char* error_message = NULL;

    // open the database connection
    int result = sqlite3_open(":memory:", &db);

    if (result != SQLITE_OK)
    {
        std::cout << "Failed to connect to the database and terminated.
ERROR=" << sqlite3_errmsg(db) << std::endl;
        return -1;
    }

    std::cout << "Connected to the database." << std::endl;

    // initialize our database
    if (!initialize_database(db))
    {
        std::cout << "Database Initialization Failed. Terminating." <<
std::endl;
        return_code = -1;
    }
    else
    {
        run_queries(db);
    }

    // close the connection if opened
    if (db != NULL)
    {
```

Noncompliant Code

```
        sqlite3_close(db);
    }

    return return_code;
}
```

Compliant Code

The system does not contain a method that tests for SQL Injection vulnerabilities.
N/A

Principle 8: Practice Defense in Depth
Adds layer of security to protect the program from one of the most common cyberattacks, the SQL injection.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	18	1

Automation

Tool	Version	Checker	Description Tool
The Checker Framework	2.1.3	Tainting Checker	Trust and security errors
CodeSonar	7.4p0	JAVA.IO.INJ.SQL	SQL injection
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

Coding Standard 5: Memory Protection

Coding Standard	Label	Name of Standard
Memory Protection	[STD-005-CPP]	The system's memory is protected so that no residual data is left, forgotten, or not handled at all.

Noncompliant Code

The system properly handles a user transaction and then deletes it so it is not repeated after the fact.

```
int* userTransaction = new int[2] {0, 1200};
…
delete[] userTransaction;
```

Compliant Code

The system fails to clear a user transaction and continues to reassign new values to the same variable. May create data corruption issues.

```
int* userTransaction = new int[2] {0, 1200};
userTransaction = new int[2] {0, 5000};
userTransaction = new int[2] {1, 3400};
userTransaction = new int[2] {0, 80};
```

Principle 7: Sanitize Data Sent to Other Systems
Helps retain and secure data in use, transit, and rest.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
High	Likely	Medium	18	1

Automation

Tool	Version	Checker	Description Tool
Helix QAC	2023.3	DF4761, DF4762, DF4766, DF4767	
Parasoft C/C++ test	2023.1	CERT_CPP-MEM53-a	Do not invoke malloc/realloc for objects having constructors
Polyspace Bug Finder	R2023b	CERT C++:MEM53-CPP	Checks for objects allocated but not initialized
PVS-Studio	7.26	V630, V749	

Coding Standard 6: Assertions

Coding Standard	Label	Name of Standard
Assertions	[STD-006-CPP]	Assertions help detect bugs in the program based on Boolean values. If it is assumed true, the system will point out a change in the assertion so developers can address the situation.

Noncompliant Code

System initializes a variable and asserts a value for verification in the remainder of the program.

```
int main() {
     int loginSuccess = 0;
     …
     assert(loginSuccess = 1);
     …
     return 0;
}
```

Compliant Code

System initializes a variable but does not assert a value for checking later in the program

```
int main() {
     int loginSuccess = 0;
     …
     …
     …
     return 0;
}
```

Principle 4: Keep It Simple
Provides quick checks and validations for other methods. Prevents verbose code.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	High	1	3

Automation

Tool	Version	Checker	Description Tool
Astree	22.10	unhandled-throw-noexcept	Partially checked
Axivion Bauhaus Suite	7.2.0	CertC++-ERR55	
CodeSonar	7.4p0	LANG.FUNCS.ASSERTS	Not enough assertions
Parasoft C/C++test	2023.1	CERT_C-MSC11-a	Assert liberally to document internal assumptions and invariants

Coding Standard 7: Exceptions

Coding Standard	Label	Exceptions
Exceptions	[STD-007-CPP]	Allows the program to use handlers to catch any exceptional circumstances while the system is running.

Noncompliant Code

The program contains decision branching that allows the system to react to certain exceptions the user reaches.

```
int main() {
     try
     {
          throw loginException
     }
     catch (int loginFailure) {
          cout << "Error: Login credentials are invalid.";
     }
}
```

Compliant Code

The program does not have a way to handle exceptions or bugs in the program if they occur.

```
int main() {
     displayLogin();
     verifyLogin();
     displayMenu();
     …
     …
     …
}
```

Principle 2: Heed Compiler Warnings
Developers can notate which errors the IDE catches, as well as what the system returns when an exception is thrown during the deployment phase of the program.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Low	Likely	Low	9	2

Automation

Tool	Version	Checker	Description Tool
Astree	22.10	unhandled-throw-noexcept	Partially checked
Axivion Bauhaus Suite	7.2.0	CertC++-ERR55	
CodeSonar	7.4p0	LANG.STRUCT.EXCP.THROW	Use of throw
Helix QAC	2023.3	C++4035, C++4036, C++4632	

Coding Standard 8: Object Oriented Programming

Coding Standard	Label	Object Oriented Programming
Object Oriented Programming	[STD-008-CPP]	The system is organized into objects or modules that allow the user to safely access different areas of the program. It also allows developers to organize their application into different layers which can offer efficiency when coding securely or patching bugs.

Noncompliant Code

The application is organized into different modules by allocating different features of the application to their own function.

```
int displayLogin() {
     …
}

int verifyLogin() {
     …
}

int displayMenu() {
     …
}



int main() {
     displayLogin();
     verifyLogin();
     displayMenu();
     …
     …
     …
}
```

Compliant Code

The program is not organized and all code is maintained in a single function.

```
int main() {
     // all program code here
     …
     …
}
```

Principle 3: Architect and Design for Security Policies
Organizes the program into digestible modules which will benefit in unit testing and security implementations/dependencies.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Unlikely	Medium	4	3

Automation

Tool	Version	Checker	Description Tool
Astree	22.10	initializer-list-order	Fully checked
Axivion Bauhaus Suite	7.2.0	CertC++-OOP53	
Clang	3.9	-Wreorder	
CodeSonar	7.4p0	LANG.STRUCT.INIT.OOMI	Out of Order Member Initializers

Coding Standard 9: Input Output

Coding Standard	Label	Input Output
Input Output	[STD-009-CPP]	Proper input/output usage in a program is essential for the security of data in use, in transit, and at rest.

Noncompliant Code

The program retrieves the proper input with the corresponding variables and outputs the proper statements.

```
int main() {
     cin >> username;
     cin >> password;
     cout << "Thank you for entering your credentials.";

     // Remainder of program
     …
     …
}
```

Compliant Code

The program retrieves the proper input with the corresponding variables but does not provide an output to notify the user that the credentials have been received.

```
int main() {
     cin >> username;
     cin >> password;

     // Remainder of program
     …
     …
}
```

Principle 1: Validate Input Data
Ensures data is secure in use, transit, and rest.

Threat Level

Severity	Likelihood	Remediation Cost	Priority	Level
Medium	Unlikely	Medium	4	3

Automation

Tool	Version	Checker	Description Tool
CodeSonar	7.4p0	ALLOC.LEAK	Leak
Helix QAC	2023.3	DF4786, DF4787, DF4788	
Klocwork	2023.3	RH.LEAK	
Parasoft C/C++test	2023.1	CERT_CPP-FIOS1-a	Ensure resources are freed

Coding Standard 10: Declarations and Initialization

Coding Standard	Label	Declarations and Initialization
Declarations and Initialization	[STD-010-CPP]	Allows the program to declare variables and initialize them for use later in the program.

Noncompliant Code

The program initializes the username and password variables for use when checking login credentials of the user.

```
int main() {
     string username;
     string password;

     cin >> username;
     cin >> password;
     cout << "Thank you for entering your credentials.";

     // Remainder of program
     …
     …
}
```

Compliant Code

The program does not initialize the username and password variables for use when checking login credentials of the user. The system will run into an error which will not allow the program to proceed since the data type was not established, nor was the variable.

```
int main() {
     cin >> username;
     cin >> password;
     cout << "Thank you for entering your credentials.";

     verifyLogin(username, password);

     // Remainder of the program
     …
     …
}
```

Principle 4: Keep It Simple
Variables are properly initialized, typed, and named at the beginning of the file or method in order for proper use, storage, and management of data later in the program.

Threat Level

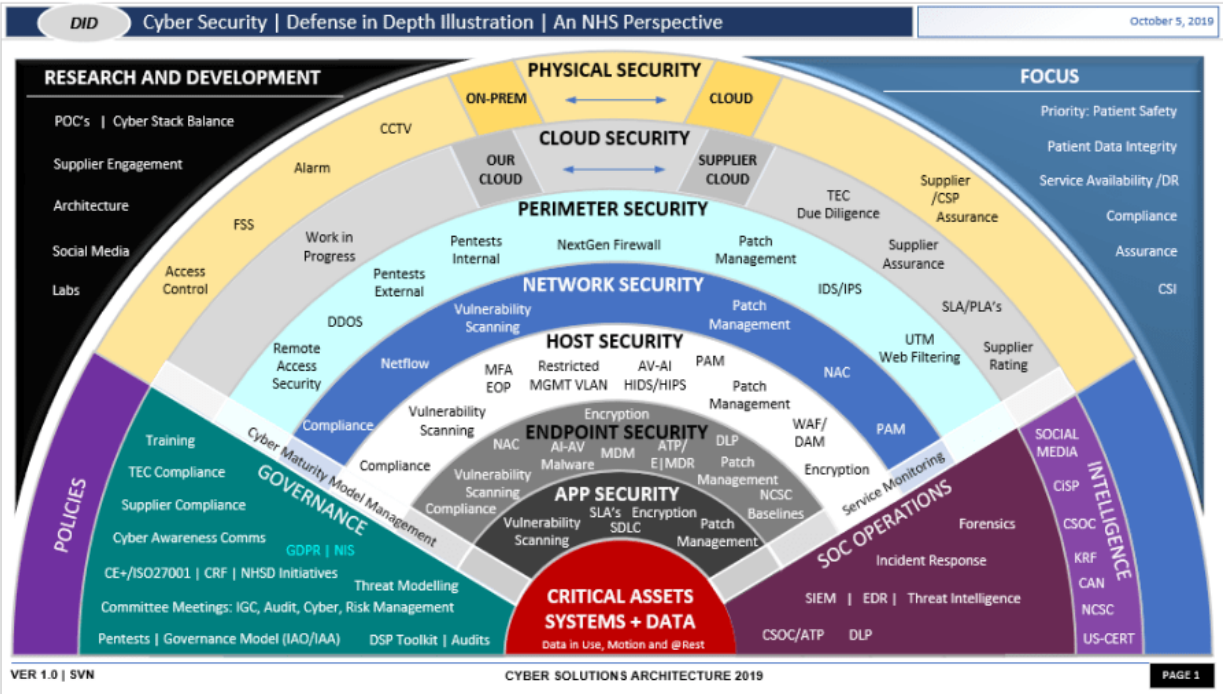
Severity	Likelihood	Remediation Cost	Priority	Level
Low	Unlikely	Medium	2	3

Automation

Tool	Version	Checker	Description Tool
CodeSonar	7.4p0	LANG.STRUCT.DECL.FNEST	Nested Function Declaration
Helix QAC	2023.3	C++1109, C++2510	
Klocwork	2023.3	CERT.DCL.AMBIGUOUS_DECL	
LDRA tool suite	9.7.1	296 S	Partially implemented

Defense-in-Depth Illustration

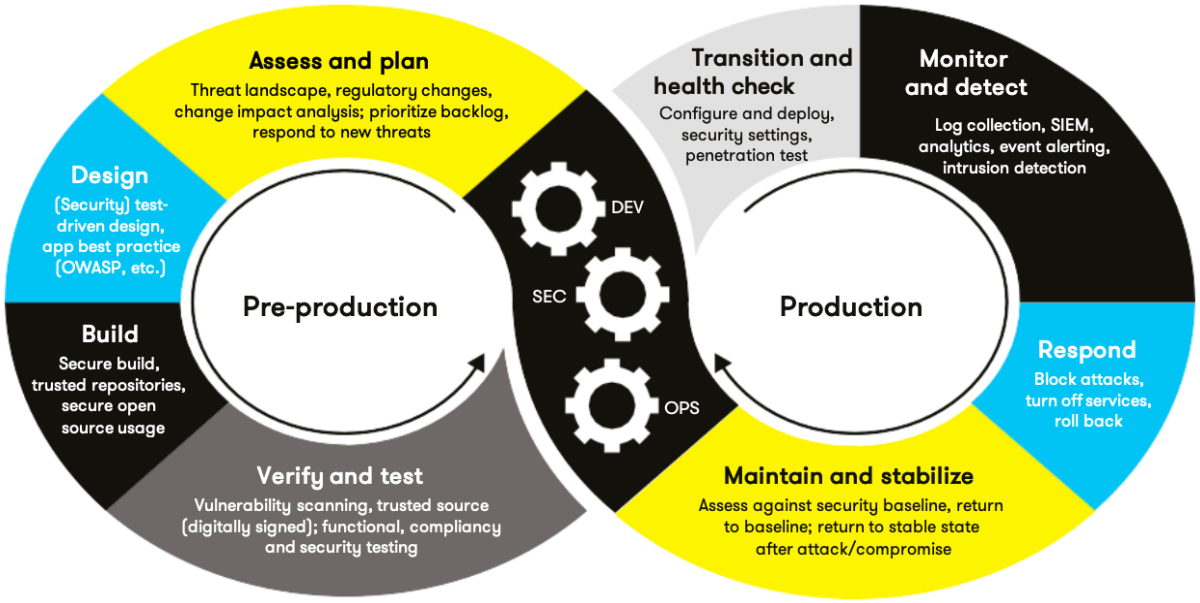
This illustration provides a visual representation of the defense-in-depth best practice of layered security.



Project One

Automation

Automation allows certain areas of development to be streamlined. Dependencies are tools which help development check, detect, and advise on vulnerable areas that can be fixed for better program security. Automation tools can continue to run during the deployment phase of the software development lifecycle as well.



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. Follow up with the project manager

and head of cybersecurity to see more on standards and policies. Core DevOps practices of Green Pace can be referred to in the Ten Core Security Principles on page 3. The diagram above shows the DevSecOps process and how a program is built with security in mind, from planning to start to finish.

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	Unlikely	Medium	2	3
STD-002-CPP	High	Likely	Medium	18	1
STD-003-CPP	Low	Likely	Low	9	2
STD-004-CPP	High	Likely	Medium	18	1
STD-005-CPP	High	Likely	Medium	18	1
STD-006-CPP	Low	Unlikely	High	1	3
STD-007-CPP	Low	Likely	Low	9	2
STD-008-CPP	Medium	Unlikely	Medium	4	3
STD-009-CPP	Medium	Unlikely	Medium	4	3
STD-010-CPP	Low	Unlikely	Medium	2	3

Create Policies for Encryption and Triple A

a. Encryption	Description
Encryption in rest	Secures data at rest. Produces minimal effect on input/output latency and throughput. The policy applies to protect data and make it transparent only to relevant users, applications, and services. ● Only users with keys are permitted to access data at rest ● Data is not stored for longer than needed
Encryption at flight	Secures data in transit. Produces minimal effect on input/output latency and throughput. The policy applies to protect data and make it transparent only to relevant users, applications, and services. ● Only users with keys are permitted to access data in transit ● Data is not in transit unless needed for purposes of storage or use
Encryption in use	Secures data in use. Produces minimal effect on input/output latency and throughput. The policy applies to protect data and make it transparent only to relevant users, applications, and services. ● Only users with keys are permitted to access data in use ● Data is not used or accessed unless needed and by parties with approved access

b. Triple-A Framework*	Description
Authentication	Authentication is the verification of a user, process, or device's identity. It is a prerequisite before the system grants them access to resources in a system. ● Authentication must be implemented at any access/entry point ● Multi factor authentication must be used ● Keys cannot be hard coded into the program ● Implemented in user logins
Authorization	Authorization is when a server determines if the client or relevant party has permission to use or access data. ● Account permissions must be implemented and provide various levels of access, and various levels of files accessed by users ● All accounts must possess the minimal permissions ● Permissions cannot be granted by anyone except for administrator accounts ● Only administrators can make changes to the database
Accounting	Accounting tracks and records user activities on a network or system. ● An audit log of all user activity must be made ● The audit log can only be accessed by administrator accounts ● Nobody is allowed to alter the audit log ● New users can be self-initiated

---

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	
1.2	11/12/2023	Updated coding standards and examples.	Jasmine Zeng	Green Pace
1.3	12/1/2023	Updated remainder of template.	Jasmine Zeng	Green Pace

Appendix A Lookups

Approved C/C++ Language Acronyms

Language	Acronym
C++	CPP
C	CLG
Java	JAV