**Green Pace Developer: Security Policy Guide**



Green Pace Secure Development Policy

GUIDE FOR USE

Presented by:

Marta Taylor

April 18, 2021

Contents

[Overview 2](#_Toc69643795)

[Introduction 2](#_Toc69643796)

[Threats Matrix 2](#_Toc69643797)

[Use of Automation 2](#_Toc69643798)

[Ten Core Security Principles 4](#_Toc69643799)

[C/C++ Ten Coding Standards 5](#_Toc69643800)

[Encryption Policies 6](#_Toc69643801)

[Triple-A Framework Policy - Explained 7](#_Toc69643802)

[Unit Testing 7](#_Toc69643803)

[Automation Summary 9](#_Toc69643804)

[Risks, Benefits and Conclusion 9](#_Toc69643805)

# Overview

## Introduction

The Green Pace security policy defines the core security principles, standards, and best practices required to be followed and incorporated in development at Green Pace. 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.

## Threats Matrix

This table provides a summary of all identified security risks. As can be seen, the majority of the coding standards defined by this policy are associated with high-risk threats that have severe consequences and are most likely to occur during an attack.

|  |  |  |  |
| --- | --- | --- | --- |
| **PROBABILITY** | **HARM SEVERITY** | | |
| **LOW** | **MEDIUM** | **HIGH** |
| **LIKELY** |  |  | STD-003  STD-004  STD-005  STD-006  STD-010 |
| **PROBABLE** | STD-007 |  | STD-002  STD-009 |
| **UNLIKELY** | STD-001 | STD-008 |  |

## Use of Automation

There are specific tools that may be used to automatically detect issues. There are different tools recommended for each coding standard. Some tools are more universal in nature and may apply to multiple standards. One tool, CPPCheck, can be used to analyze code for a multitude of vulnerabilities, identify exceptions and suggest mitigation measures. As can be seen in the figure below, CPPCheck identifies an out of bounds index. Accessing an index outside of the predefined range results in unexpected / uncontrolled code behavior and is thus a vulnerability (see STD-002, Principle 1).

Graphical user interface, text, application, email

Description automatically generated

# Ten Core Security Principles

| **Principles** | **Description / Coding Standards** |
| --- | --- |
| 1. ValidateInput Data | All input shall be validated to prevent software vulnerabilities. All external data sources shall be considered unsafe including command line arguments, network interfaces, environmental variables and user-controlled files, for example.  STD-001-CLG  STD-002-CPP  STD-003-CPP  STD-004-CLG  STD-010-CLG |
| 1. Heed Compiler Warnings | The highest compiler warning levels shall be used to compile code. All compiler warnings shall be addressed and eliminated by modifying code and using static and dynamic testing tools to detect and remove any additional security flaws.  STD-001-CLG  STD-002-CPP  STD-005-CPP |
| 1. Architect and Design for Security Policies | Design and build software that enforces and heeds security policies. An example might be using private methods or classes.  STD-004-CLG  STD-007-CPP  STD-009-CPP |
| 1. Keep It Simple | Code design shall be kept as small and simple as possible. The simpler the code, the less likelihood for errors and oversights.  STD-001-CLG  STD-009-CPP |
| 1. Default Deny | Access decisions shall be based on permission – not exclusion. Users, for example, shall not obtain access to a system without proper user credentials. |
| 1. Adhere to the Principle of Least Privilege | Each process shall execute with the least amount of privilege needed to perform that function. Least privilege pertains to users, programs, processes, networks, databases or entire systems. |
| 1. Sanitize Data Sent to Other Systems | Data shall be sanitized PRIOR to passing it to another system / subsystem. An example is removing SQL injection from input prior to passing it as an argument to another method.  STD-003-CPP  STD-004-CLG |
| 1. Practice Defense in Depth | Systems shall be protected in layers. Risk shall be managed with multiple strategies to prevent exploits of vulnerabilities.  STD-001-CLG  STD-002-CPP  STD-003-CPP  STD-008-CPP  STD-009-CPP |
| 1. Use Effective Quality Assurance Techniques | Vulnerabilities can be identified and eliminated using good quality assurance techniques such as third-party security reviews.  STD-005-CPP  STD-006-CPP  STD-007-CPP  STD-010-CLG |
| 1. Adopt a Secure Coding Standard | A secure coding standard shall be developed for each individual programming language and platform as different languages come with different vulnerabilities.  STD-008-CPP |

Source: <https://wiki.sei.cmu.edu/confluence/display/seccode/Top+10+Secure+Coding+Practices>

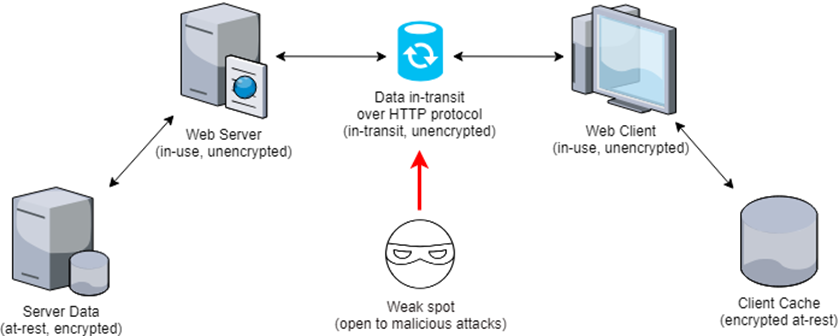
# C/C++ Ten Coding Standards

The ten coding standards have been organized by Level, where the Level summarizes the severity of the consequences of NOT implementing the standard; the likelihood that an exploitable vulnerability will occur by NOT implementing the standard; and the expense associated with complying with the standard. Thus, it is more imperative to comply with the Level 1 standards as compared to complying with the Level 3 standards.

| **Rule** | **Severity** | **Likelihood** | **Remediation Cost** | **Priority** | **Level** |
| --- | --- | --- | --- | --- | --- |
| STD-002-CPP | HIGH | PROBABLE | MEDIUM | 12 | 1 |
| STD-003-CPP | HIGH | LIKELY | MEDIUM | 18 | 1 |
| STD-004-CLG | HIGH | LIKELY | MEDIUM | 18 | 1 |
| STD-005-CPP | HIGH | LIKELY | MEDIUM | 18 | 1 |
| STD-009-CPP | HIGH | PROBABLE | HIGH | 6 | 2 |
| STD-010-CLG | HIGH | LIKELY | HIGH | 9 | 2 |
| STD-006-CPP | HIGH | LIKELY | HIGH | 9 | 2 |
| STD-001-CLG | LOW | UNLIKELY | LOW | 3 | 3 |
| STD-008-CPP | MEDIUM | UNLIKELY | MEDIUM | 4 | 3 |
| STD-007-CPP | LOW | PROBABLE | MEDIUM | 4 | 3 |

# Encryption Policies

| **Encryption** | **How and Why the Policy Applies.** |
| --- | --- |
| Encryption in rest | Protects data where it is stored (eg: on a computer/phone, a database, in cloud). This may include physical sources such as hard drives, or logical sources such as databases or cloud assets. There are several options for encryption tools, such as VeraCrypt, AxCrypt or InnoDB tablespace encryption. These tools are necessary to protect data from being physically stolen, logically stolen, or otherwise be breached. |
| Encryption at flight | Protects data as it is moved from one location to another (eg: sending an email, browsing the internet). Email encryption tools such as S/MIME or PGP shall be used for email transmission. Web traffic shall only be sent over a Secure Sockets Layer (SSL) such as Transport Layer Security (TLS) by obtaining a SSL/TLS HTTPS certificate from authorities such as GoDaddy or DigiCert. Refer to Table 1 below for suggested secure network protocols. Using these tools prevents network layer attacks, such as eavesdropping, and tampering-based attacks, such as third-party communication hijacking. |
| Encryption in use | Protects data as it is being created, edited, accessed, processed, or viewed. This state occurs in between the at-rest and at-flight states when, say for instance, accessing a website on a server, or whenever the CPU is in use processing applications. It is important to encrypt data-in-use because memory can be hacked, and encryption keys for data-at-rest may be exposed. CPU manufacturer AMD offers full memory encryption, called Secure Memory Encryption (SME), and Intel offers Total Memory Encryption (TME) in order to protect CPU based key storage. There are also cryptographic tools that can be used to protect data during computation. |



Source: <https://www.ryadel.com/en/data-encryption-in-transit-at-rest-definitions-best-practices-tutorial-guide/>

# Triple-A Framework Policy - Explained

| **Triple-A Framework** | **How and Why This Policy Applies** |
| --- | --- |
| Authentication | Who are you? Authentication verifies a user’s identity credentials. This can be achieved in several ways, including username/password verification, single sign-on (SSO) systems, biometrics, I and/or digital certificates, for example. All users, new and existing, shall be verified. Identity theft and unauthorized system access can result if user credentials are not authenticated. |
| Authorization | What can you use? Authorization is set for each existing and new user and defines the level of access to files, directories and/or applications for each individual user. Each user or group shall be permitted to have read, write and/or execution permission, depending on the user’s role within an organization. For example, a data entry clerk may have read / write access to view or edit a particular file while a manager may warrant administrative permissions. |
| Accounting | What happened and when? Accounting refers to records, or log files, which detail things like user logins; new user profile creations; file access events; database updates; data transfers or access. All actions are date and timestamped as a record of occurrence. Usage information can track events such as authorization or resource utilization which can be used in turn for system wide planning. Accounting tracks who is doing what at all times. |

Source: <https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting>

# Unit Testing

Unit testing is used to verify and validate individual components of an application and ensure that the app will respond as intended or expected. Unit testing at Green Pace is accomplished by using the Google Test unit test execution environment, and [Google’s GitHub Primer](https://github.com/google/googletest/blob/master/docs/primer.md) page to reference the ASSERT\_ / EXPECT\_ functions within Google Test. Tests shall be independent from one another, repeatable, portable, and reusable. Unit testing can be used to test methods, user input, range boundaries, variable (in)equalities, or array contents for example.

An example of a unit test which verifies that the actual size of a vector is less than the maximum allowed vector size, is shown below. The Google Test functions EXPECT\_TRUE and EXPECT\_EQ are used to verify that the actual contents of the vector are the same as the expected contents.

Text

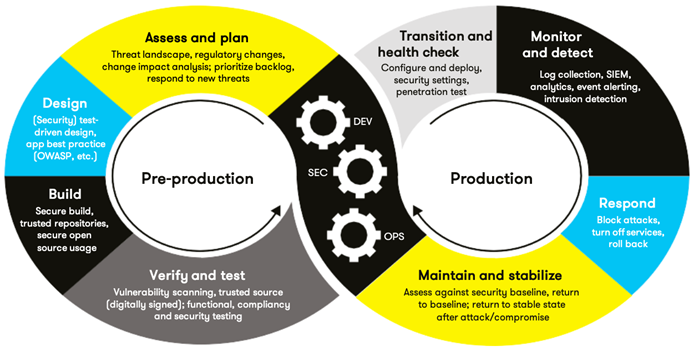
Description automatically generated

Another example of how a unit test can be used is by verifying that an exception is thrown when known error occurs. Here, an iterator range is greater than the number of items stored in the vector. The error message thrown describes the error so that the user is aware of the nature of the bug identified. The assertion tests that the exception is thrown, and the error message is displayed as expected.

Text

Description automatically generated

# Automation Summary

Automation will be used for the enforcement of and compliance to the standards defined in this policy. DevOps transforms to DevSecOps by integrating security measures into each step of the DevOps toolchain. Within the pre-production phases, threat modeling and security tool training and selections are added to the “Assess and Plan” segment. In the “Design” and “Build” phases, IDE security is addressed. Static application testing and automated security scans are added to the “Verify & Test” phase along with unit, integration and other tests.

The Google Test unit tests shall be grouped together to form test “suites” which are collections of tests that can be automated and run daily. There are a host of other tools which can be used to identify compilation errors, warnings, and/or style suggestions. CPPCheck is one such tool which can be used for static analysis, and all resulting notes made by CPPCheck shall be addressed prior to releasing the code. It is beneficial to run static code through at least two analysis tools, as there is no perfect tool that can catch all bugs, and some tools catch more bugs than others.

# Risks, Benefits and Conclusion

The protection of software code is no longer an option in this day and age of cybercrimes – it is mandatory. Whether an attacker seeks to completely disable an application (2017 WannaCry ransomware attack), or the motive is theft of confidential data (2017 Equifax breach), or to take control over the outcome of an entire election (2016 US Presidential Election), the benefits of testing early and testing often far outweigh the huge negative consequences of neglecting compliance with this policy, the coding standards contained herein, as well as secure coding best practices.

The threats of today are not necessarily the threats of tomorrow. Secure coding is not something that can be done once and put on a shelf. This security policy shall be a living document, and shall be reviewed and updated at least annually, or when new threats are identified. The coding standards incorporated into this presentation are a mere start to protection against exploitable vulnerabilities. Additional standards shall be added to address and incorporate compliance with all ten principles. It is also recommended that Green Pace hire a white hat cyber security firm to test all applications for vulnerabilities. The cost of protection pales in comparison to the risk of exploitation.

Taking the extra step of protecting code from the start, implementing defense in depth and incorporating daily testing routines into everyday practice, DevOps can be transformed into a DevSecOps process here at Green Pace.