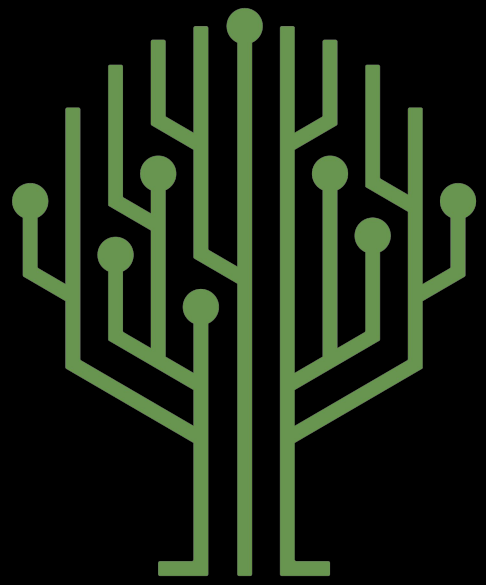
Green Pace

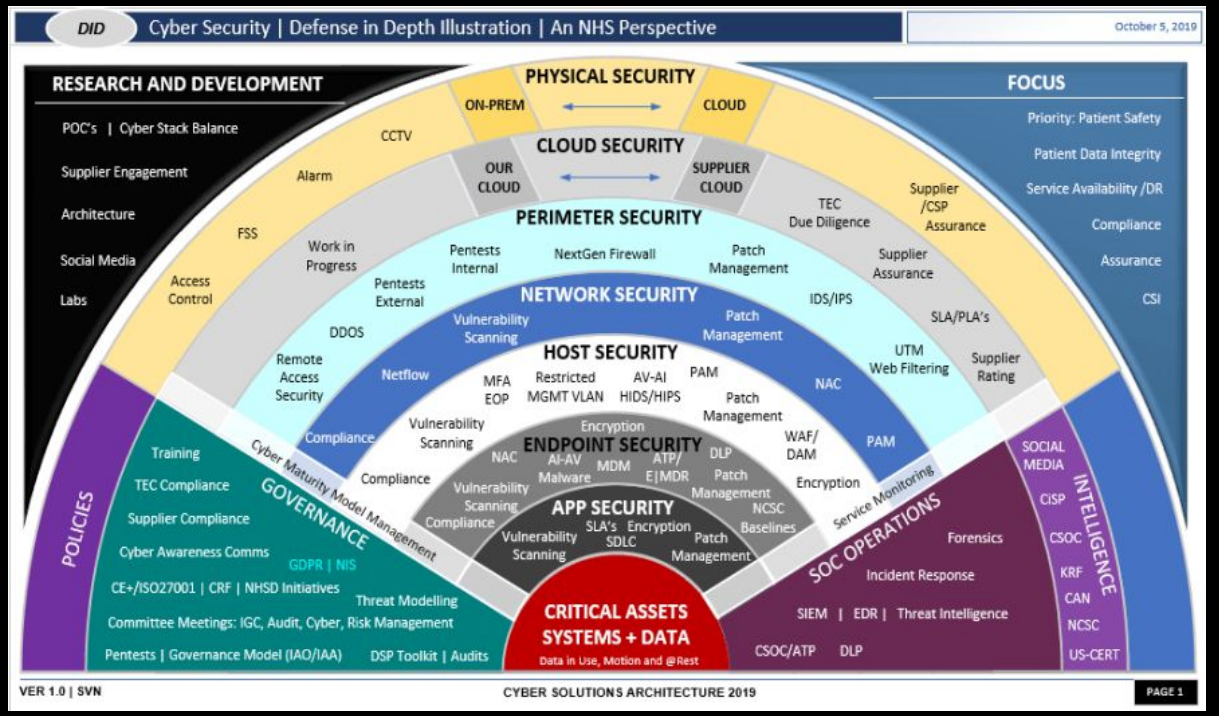
Security Policy Presentation

Developer: Brad Jackson



OVERVIEW: DEFENSE IN DEPTH

Software development at Green Pace lacked a deep-seated implementation of security principles. These principles are required to ensure application development is uniformly defined, implemented, governed, and maintained over time to enable defense in depth.



Green Pace

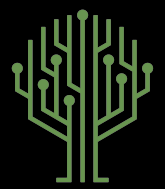
THREATS MATRIX

Likely (Short Term) STD-005-CPP STD-007-CPP STD-010-CPP	**Priority** STD-002-CPP STD-004-CPP
Low priority STD-001-CPP STD-006-CPP STD-008-CPP STD-009-CPP	**Unlikely (Long Term)** STD-003-CPP

Green Pace

10 PRINCIPLES

- Validate Input Data
  - STD-004-CPP
  - STD-006-CPP
- Heed Compiler Warnings
  - STD-001-CPP
  - STD-002-CPP
  - STD-005-CPP
  - STD-006-CPP
  - STD-008-CPP
- Architect and Design for Security Policies
  - STD-007-CPP
- Keep It Simple
  - STD-008-CPP
- Default Deny
  - STD-009-CPP
- Adhere to the Principle of Least Privilege
  - STD-009-CPP
- Sanitize Data Sent to Other Systems
  - STD-004-CPP
  - STD-007-CPP
- Practice Defense in Depth
- Use Effective Quality Assurance Techniques
  - STD-003-CPP
  - STD-004-CPP
  - STD-005-CPP
  - STD-006-CPP
  - STD-007-CPP
  - STD-008-CPP
  - STD-010-CPP
- Adopt a Secure Coding Standard
  - STD-004-CPP
  - STD-009-CPP
  - STD-010-CPP



CODING STANDARDS

- Higher Priority (P18) > Lower Priority (P1)
- Based on Severity, Likelihood and Remediation Cost

Rule	Severity	Likelihood	Remediation Cost	Priority	Level
STD-001-CPP	Low	Unlikely	Low	P3	L3
STD-002-CPP	High	Probable	Medium	P12	L1
STD-003-CPP	High	Unlikely	Medium	P6	L2
STD-004-CPP	High	Probable	Medium	P12	L1
STD-005-CPP	High	Likely	Medium	P18	L1
STD-006-CPP	Low	Unlikely	High	P1	L3
STD-007-CPP	Low	Likely	Low	P9	L2
STD-008-CPP	Medium	Unlikely	Medium	P4	L3
STD-009-CPP	Medium	Unlikely	Medium	P4	L3
STD-010-CPP	Low	Likely	Low	P9	L2

Green Pace

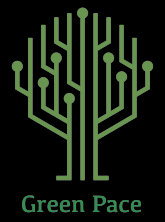
ENCRYPTION POLICIES

Encryption	Summary
Encryption in rest	Encryption in rest protects stored data ranging from local hardware (HDD, SSD,NVMe) storage to cloud assets. Encryption is a requirement for sensitive data to prevent potential attackers from seizing digital assets or private information.
Encryption at flight	Encryption at flight (or encryption in transit) pertains to protecting data transferred between systems. Data is protected in transit by scrambling the data so that, if intercepted, bad actors cannot easily consume it. This is usually implemented using SSL or TLS.
Encryption in use	Encryption in use protects data that is created, edited, or otherwise defined as in-use. Encrypting data in memory slows data processing; however, if an application is dealing with highly sensitive data, this precaution is necessary.

Green Pace

TRIPLE-A POLICIES

Triple-A Framework	Summary
Authentication	Authentication means validating a user's identity. The simplest form of authentication is using a username and password. In recent years, and depending on the sensitivity of an organization's information, multiple factors of authentication are used to validate a user's identity. Time based one-time passwords, access tokens, and biometric credentials are just a few ways of establishing known identity. Authentication is vital to ensure the security of business-critical information.
Authorization	Authorization specifies the access rights and privileges of a user, and are an important part of information security. Authorization determines what a user can and cannot access. Adopting least privilege limits possible leaks of information outside of your organization. New users should only have access to resources they need to access according to their job description or teammates' access. Users should not have access beyond what is required by their role. The same applies to service accounts and system users. Hijacked accounts become less of a threat when they are unable to access system vitals.
Accounting	Accounting or auditability means ensuring proper and accurate logging is in place. Sensitive data and systems should have a ledger of transactions and user access timestamps. In the event of an incident, the incident response team will then quickly be able to quarantine any related users or applications to prevent a more critical issue. The easiest and most common practice of this is within databases, where virtually all access and transaction can be logged. In the event of a breach, teams can determine exactly who accessed the data and when to isolate it and begin triage.



Unit Testing for String Correctness

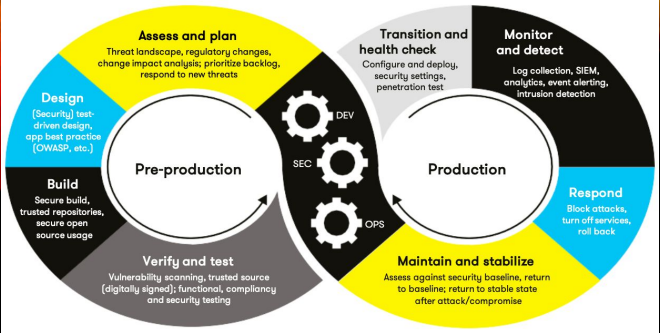
Coding Standard 3 - String Correctness - STD-003-CPP - Range check element access.

```cpp
// 👇 Test (negative) that calling reserve with argument greater than
// 👇 max size throws a length_error exception
TEST_F(CollectionTest, ReserveGreaterThanMaxSizeThrowsException)
{
    // 👇 size is 0
    ASSERT_EQ(collection->size(), 0);

    bool inLength = true;

    // 👇 catch reserve with argument greater than max_size
    try
    {
        collection->reserve(collection->max_size() + 1);
    }
    catch (std::length_error &)
    {
        inLength = false;
    }
    ASSERT_TRUE(inLength);
}
```

```cpp
// 👇 Test to verify the std::out_of_range exception
// 👇 is thrown when calling at() with an index out of bounds
TEST_F(CollectionTest, ExceptionThrownIfOutOfBounds)
{
    // 👇 size is 0
    ASSERT_EQ(collection->size(), 0);
    ASSERT_EQ(collection->capacity(), 0);

    // 👇 add one entry
    add_entries(1);

    bool inRange = true;

    // 👇 exception is thrown when accessing index 1
    try
    {
        collection->at(1);
    }
    catch (std::out_of_range &)
    {
        inRange = false;
    }

    ASSERT_TRUE(inRange);
}
```

```cpp
// 👇 Test to verify adding five values to collection
TEST_F(CollectionTest, CanAddFiveValuesToVector)
{
    // 👇 is the collection empty?
    ASSERT_TRUE(collection->empty());

    // 👇 if empty, the size must be 0
    ASSERT_EQ(collection->size(), 0);

    add_entries(5);

    // 👇 is the collection still empty?
    ASSERT_FALSE(collection->empty());

    // 👇 if not empty, what must the size be?
    ASSERT_EQ(collection->size(), 5);
}
```

Green Pace



AUTOMATION SUMMARY

Automation should be a part of the build and verify phases of pre-production. Static code scans should be required as well as validation from the development team that testing has been completed. In terms of automation within the CI/CD pipeline, GitOps can be utilized to implement automated code scanning tools when commits are made to internal repositories. Automated gatekeeping processes should prevent applications with known vulnerabilities from being deployed in production until the findings are remediated.

Green Pace

TOOLS

- **The CI/CD pipeline should be initiated by a Git hook. Scans will be initiated as a result of the webhook. This will add time to the build process; however, it is a fundamental way to assure compliance with common coding standards.**
- **Checkmarx**: a Static Application Security Testing (SAST) tool which provides fast and accurate incremental or full scans and gives you the flexibility, accuracy, integrations, and coverage to secure your applications
- **Cppcheck**: a static analysis tool for C/C++ code. It provides unique code analysis to detect bugs and focuses on detecting undefined behaviour and dangerous coding constructs.
- **Clang**: a compiler front end for the C, C++, Objective-C, and Objective-C++ programming languages, as well as the OpenMP, OpenCL, RenderScript, CUDA, and HIP frameworks.

Green Pace

RISKS AND BENEFITS

- Risk of waiting
  - Code debt
  - Data loss/Loss of customer trust
  - Financial cost (compliance)

- Benefits of early action
  - Prevent possible damage by prevention
  - Uniform DiD
  - Reduce incident response overhead

- Example
  - 2012 Sony Pictures Europe Breach
  - > 1 million users affected
  - SQL injection attack

Green Pace

RECOMMENDATIONS

- Policy is a living document: It should be subjected to regular reviews and updates as gaps are identified.
- Third-party validation: Annual checks from an outside source, such as a white hat security firm, will help to give security a real-world test and find possible vulnerabilities.
- Create a roadmap for implementing the policies as discussed ASAP

Green Pace

CONCLUSIONS

- The phrase “don’t leave security to the end” means that systems should be designed and implemented with security in mind. Security is often overlooked and is acknowledged to be a very complex ecosystem; however, there are well-established and simple guidelines for best practice when it comes to secure architecting and coding. At the lowest level, preventing threats begins at securing business-critical data.

Green Pace

REFERENCES

Checkmarx Ltd. (2022, November 23). CHECKMARX SAST: SCAN WITH EASE AT THE SOURCE CODE LEVEL. Checkmarx.com. Retrieved February 19, 2023, from https://checkmarx.com/cxsast-source-code-scanning

LLVM Developer Group. (n.d.). Getting started: Building and running clang. Clang. Retrieved February 19, 2023, from https://clang.llvm.org/get_started.html

Marjamäki, D. (n.d.). Cppcheck - A tool for static C/C++ code analysis. Cppcheck. Retrieved February 19, 2023, from https://cppcheck.sourceforge.io/

Green Pace