Green Pace

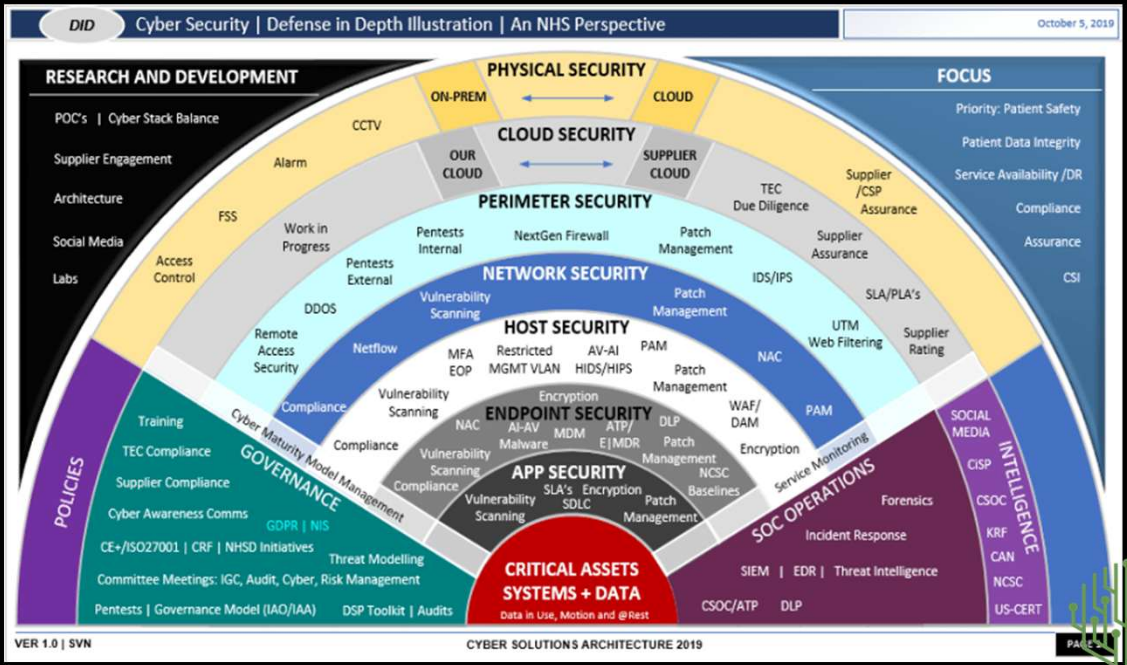
Security Policy Presentation

Developer: *Jerry Vasquez*

Green Pace

OVERVIEW: DEFENSE IN DEPTH

Defense-in-depth is a modern strategy involving utilizing multiple security layers to protect data, networks, and systems from cyber attacks. This policy aims to follow defense-in-depth by proposing various security measures at many different levels.



Green Pace

THREATS MATRIX

[Populate the Threats Matrix table and provide explanations to summarize of all of your security risks.]

Likely [Insert text here.]	Priority [Insert text here.]
Low priority [Insert text here.]	Unlikely [Insert text here.]

Green Pace

10 PRINCIPLES

1. Validate Input Data
2. Heed Compiler Warnings
3. Architect and Design for Security
4. Keep It Simple
5. Default Deny
6. Adhere to the Principle of Least Privileges
7. Sanitize Data Sent to Other Systems
8. Practice Defense in Depth
9. Use Effective Quality Assurance Techniques
10. Adopt a Secure Coding Standard

Green Pace

CODING STANDARDS

1. Implement prepared statements to prevent SQL injection (STD-004-CPP)
2. Do not access freed memory (STD-005-CPP)
3. Properly deallocate dynamically allocated memory (STD-008-CPP)
4. Ensure that operations on signed integers doe not result in overflow (STD-002-CPP)
5. Ensure strings have adequate storage to avoid risking buffer overflow (STD-003-CPP)
6. Do not mix signed and unsigned integers type (STD-001-CPP)
7. Close files when they are not needed (STD-010-CPP)
8. Handle all exceptions (STD-007-CPP)
9. Use static assertions to test the value of a constant expression (STD-006-CPP)
10. Value-returning functions must return a value from all exit paths (STD-009-CPP)

Green Pace

ENCRYPTION POLICIES

- **Encryption at Rest** – Aims to protect data while in storage. To apply this, consider full disk encryption of all data kept in storage. Encryption at rest can be beneficial if an attacker gains unauthorized access to the network. The data that the attacker will try to reach will be encrypted and unreadable.

- **Encryption in Flight** – Aims to protect data as it moves from one source to another. Encryption in flight can be achieved by implementing TLS/SSL to establish an encrypted connection between client and server thereby protecting data during transport. Encryption in flight is important to prevent bad actors from intercepting sensitive data during data transmission

- **Encryption in Use** – Aims to protect data being processed, read, or modified by employees, clients, or system applications. Encryption in use can be applied by using trusted execution environments.

Green Pace

TRIPLE-A POLICIES

- **Authentication** – Serves to verify the identity of the user trying to access the system. Authentication plays a role during user logins where the user must provide their user credentials that identifies their account. Authentication is necessary to prevent individuals with malicious intent from gaining unauthorized access to another's user account.

- **Authorization** – Serves to manage and enforce permissions regarding what a user can access and what operations they are allowed to perform. Authorization is used to determine the user levels of access. Authorization is necessary to ensure the users only have access to resources that they need.

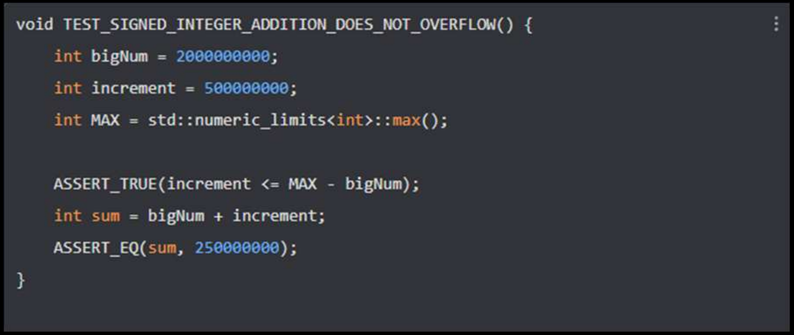
- **Accounting** – Serves to track user activity within the system. Accounting is applied to maintain transparency for which users made changes to the database, accessed specific files, etc. Accounting is important for holding individual users accountable for their actions and user activity analysis.

Green Pace

Unit Testing

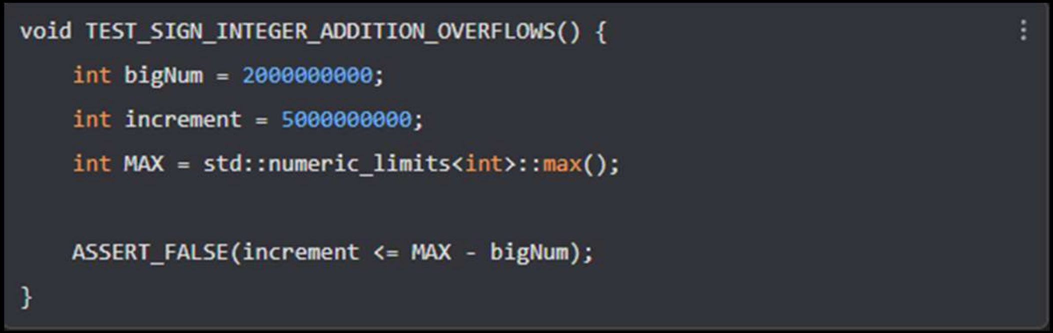
Testing Numeric Overflow

Positive Testing



```
void TEST_SIGNED_INTEGER_ADDITION_DOES_NOT_OVERFLOW() {
    int bigNum = 2000000000;
    int increment = 500000000;
    int MAX = std::numeric_limits<int>::max();

    ASSERT_TRUE(increment <= MAX - bigNum);
    int sum = bigNum + increment;
    ASSERT_EQ(sum, 250000000);
}
```

Negative Testing



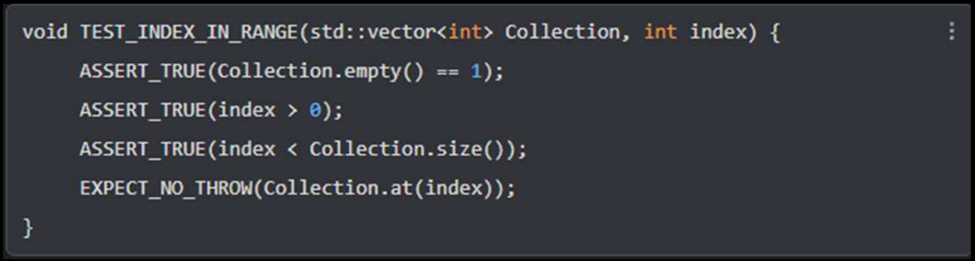
```
void TEST_SIGN_INTEGER_ADDITION_OVERFLOWS() {
    int bigNum = 2000000000;
    int increment = 5000000000;
    int MAX = std::numeric_limits<int>::max();

    ASSERT_FALSE(increment <= MAX - bigNum);
}
```

Green Pace

Unit Testing

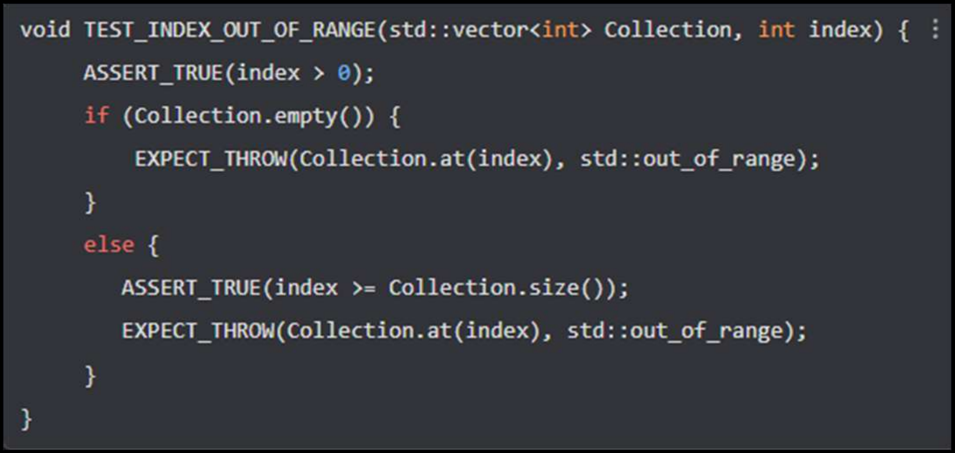
Testing Out of Range

Positive Testing



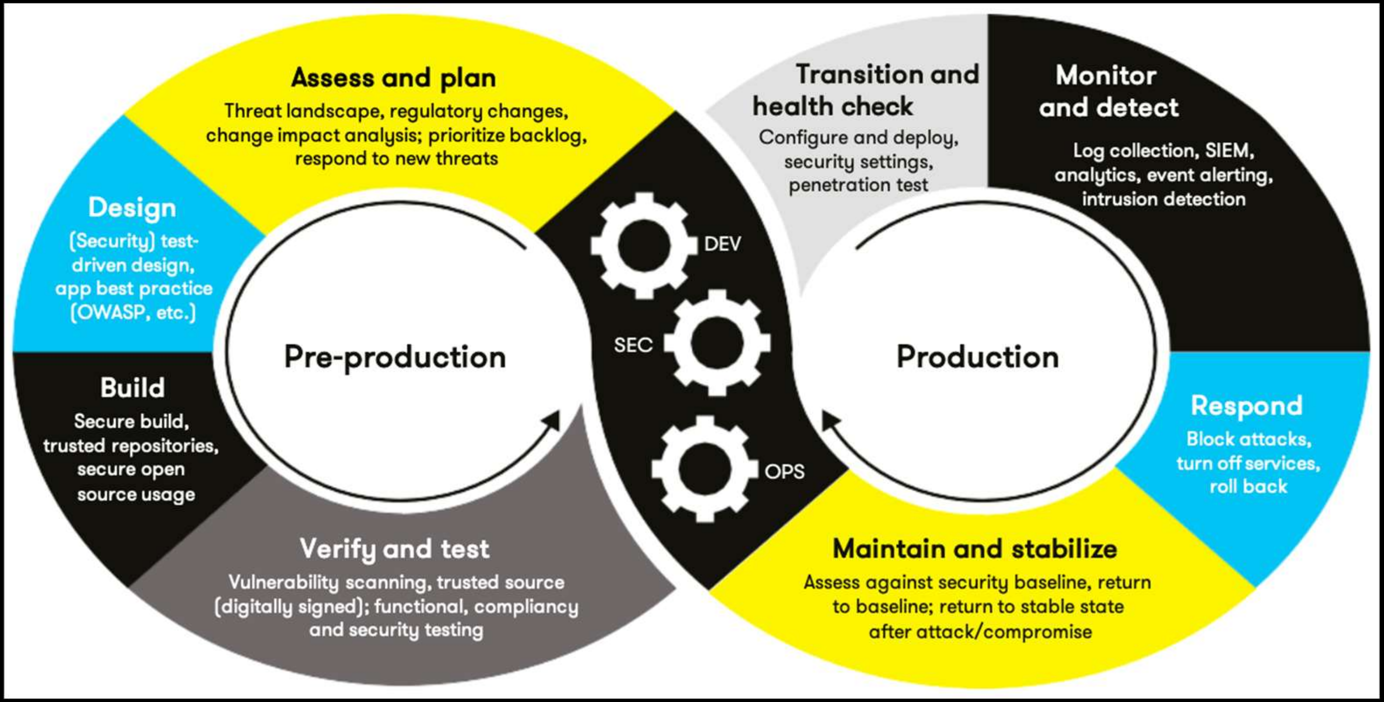
```
void TEST_INDEX_IN_RANGE(std::vector<int> Collection, int index) {
    ASSERT_TRUE(Collection.empty() == 1);
    ASSERT_TRUE(index > 0);
    ASSERT_TRUE(index < Collection.size());
    EXPECT_NO_THROW(Collection.at(index));
}
```

Negative Testing



```
void TEST_INDEX_OUT_OF_RANGE(std::vector<int> Collection, int index) {
    ASSERT_TRUE(index > 0);
    if (Collection.empty()) {
        EXPECT_THROW(Collection.at(index), std::out_of_range);
    }
    else {
        ASSERT_TRUE(index >= Collection.size());
        EXPECT_THROW(Collection.at(index), std::out_of_range);
    }
}
```

AUTOMATION SUMMARY


Assess and plan
Threat landscape, regulatory changes, change impact analysis; prioritize backlog, respond to new threats
Design
(Security) test-driven design, app best practice (OWASP, etc.)
Build
Secure build, trusted repositories, secure open source usage
Verify and test
Vulnerability scanning, trusted source (digitally signed); functional, compliancy and security testing
Pre-production
DEV
SEC
OPS
Transition and health check
Configure and deploy, security settings, penetration test
Monitor and detect
Log collection, SIEM, analytics, event alerting, intrusion detection
Production
Respond
Block attacks, turn off services, roll back
Maintain and stabilize
Assess against security baseline, return to baseline; return to stable state after attack/compromise

Green Pace

TOOLS

The DevSecOps pipeline demonstrates security considerations and implementations throughout the SDLC. It also highlights the importance of considering security during preproduction and production.

Tools:

- Static analysis tools (e.g. cppcheck)
- CI/CD tools
- Unit Testing frameworks
- Logging and monitoring tools

Green Pace

RISKS AND BENEFITS

Problems of Starting Late:

- Higher cost to address vulnerabilities and bugs
- More mistakes
- The complexity of addressing the vulnerabilities grows
- More serious vulnerabilities make it into production

Benefits of Starting Early:

- Vulnerabilities and bugs can be detected and addressed early
- Resilient architecture
- Overall better code quality
- Minimized cost

Risks of Starting Early:

- Slower development
- Increased effort

Green Pace

RECOMMENDATIONS

- Adoption of multi-factor authentication
- Strict implementation of least privileges
- Implement segmentation of the network
- Continuous monitoring and validation

Green Pace

CONCLUSIONS

- Practice Defense-in-Depth
- Adoption of secure coding standards
- Implement zero trust strategies
- Security must be considered from the start

Green Pace