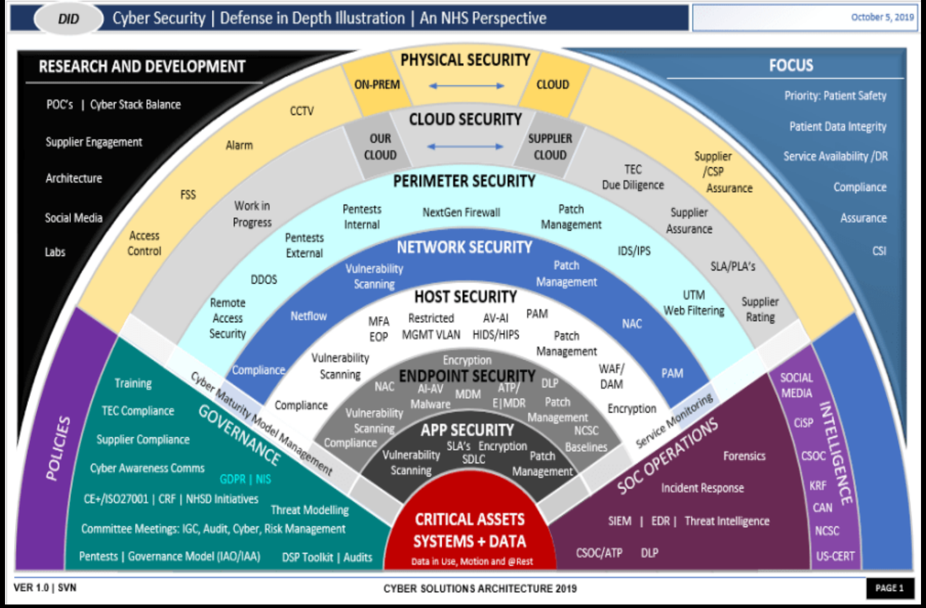
Green Pace

Security Policy Presentation
Developer: *George Kaline III*

Date:2/22/2025

OVERVIEW: DEFENSE IN DEPTH

Defense in Depth is exactly what it sounds like—a security approach with multiple layers that attackers must bypass to reach your assets. This strategy consists of 10 layers, each designed to address different aspects of security.



Green Pace

THREATS MATRIX

Likely STD-003-CPP STD-004-JAV STD-005-CPP STD-007-CPP STD-008-CLG STD-009-CPP	Priority STD-010-CPP
Low priority STD-002-CPP	Unlikely STD-001-CPP STD-006-CLG

Green Pace

10 PRINCIPLES

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

Green Pace

CODING STANDARDS

Coding Standard	Label	Name of Standard
Data Type	STD-001-CPP	Pass an object of the correct type to va_start
Data Value	STD-002-CPP	Value-returning functions must return a value from all exit paths
String Correctness	STD-003-CPP	Guarantee that storage for strings has sufficient space for character data and the null terminator
SQL Injection	STD-004-JAV	Prevent SQL injection
Memory Protection	STD-005-CPP	Do not access freed memory
Assertions	STD-006-CLG	Incorporate diagnostic tests using assertions
Exceptions	STD-007-CPP	Guarantee exception safety
Integers	STD-008-CLG	Ensure that operations on signed integers do not result in overflow
Input/Output	STD-009-CPP	Do not alternately input and output from a file stream without an intervening positioning call
Container	STD-010-CPP	Provide a valid ordering predicate

Green Pace

ENCRYPTION POLICIES

Encryption at rest	Encryption at rest protects stored data by encrypting it to prevent unauthorized access, ensuring security against theft, cyberattacks, and insider threats. It is commonly implemented through full disk encryption, file-level encryption, database encryption, or cloud storage encryption, with secure key management. This encryption is crucial for regulatory compliance and differs from encryption in transit, which secures data during transmission.
Encryption in flight	Encryption in flight, also known as encryption in transit, protects data while it is being transmitted over networks to prevent interception and unauthorized access. It is commonly implemented using protocols like TLS/SSL for web traffic, VPNs for secure remote access, and end-to-end encryption for messaging. This ensures data remains confidential and tamper-proof as it moves between systems, devices, or cloud services.
Encryption in use	Encryption in use protects data while it is actively being processed, preventing unauthorized access even when loaded in memory. It is implemented through techniques like homomorphic encryption, secure enclaves (e.g., Intel SGX, AMD SEV), and trusted execution environments (TEEs). This ensures sensitive data remains encrypted during computation, reducing risks from insider threats and memory-based attacks.

Green Pace

TRIPLE-A POLICIES

Authentication	Authentication verifies a user's identity before granting access to a system, network, or application. It uses methods like passwords, biometrics, security tokens, or multi-factor authentication (MFA) to confirm legitimacy. This step ensures that only authorized individuals can proceed to the next phases.
Authorization	Authorization determines what actions or resources a user can access after authentication. It enforces access control policies based on roles, permissions, or attributes, ensuring users can only perform approved operations. This step follows authentication and works alongside accounting to maintain security and compliance.
Accounting	Accounting tracks and logs user activities, providing an audit trail for security, compliance, and usage monitoring. It records details such as login times, accessed resources, and actions performed to detect anomalies and enforce policies. This step ensures accountability by maintaining logs for auditing, billing, and forensic investigations.

Green Pace

Unit Testing

Clearing The Collection

```
TEST_F(CollectionTest, ClearingTheCollection)
{
    add_entries(3);//set size
    ASSERT_EQ(collection->size(), 3);

    collection->clear();
    ASSERT_EQ(collection->size(), 0);
    ASSERT_TRUE(collection->empty());

}
```

```
[ RUN      ] CollectionTest.ClearingTheCollection
[       OK ] CollectionTest.ClearingTheCollection (0 ms)
```

Green Pace

Unit Testing

Verify Erase of The Collection

```
TEST_F(CollectionTest, VerifyEraseOfTheCollection)
{
    add_entries(6);//set size
    collection->erase(collection->begin(), collection->end());

    EXPECT_TRUE(collection->empty());
    ASSERT_EQ(collection->size(), 0);

}
```

```
[ RUN      ] CollectionTest.VerifyEraseOfTheCollection
[       OK ] CollectionTest.VerifyEraseOfTheCollection (0 ms)
```

Green Pace

Unit Testing

Reserve Increase Capacity of The Collection

```
TEST_F(CollectionTest, ReserveIncreaseCapacityofTheCollection)
{
    collection->reserve(6);

    add_entries(3);//set size


    EXPECT_EQ(collection->size(), 3);
    EXPECT_GE(collection->capacity(), 6);

}
```

```
[ RUN      ] CollectionTest.ReserveIncreaseCapacityofTheCollection
[       OK ] CollectionTest.ReserveIncreaseCapacityofTheCollection (0 ms)
```

Green Pace

Unit Testing

Null Or Not The Collection

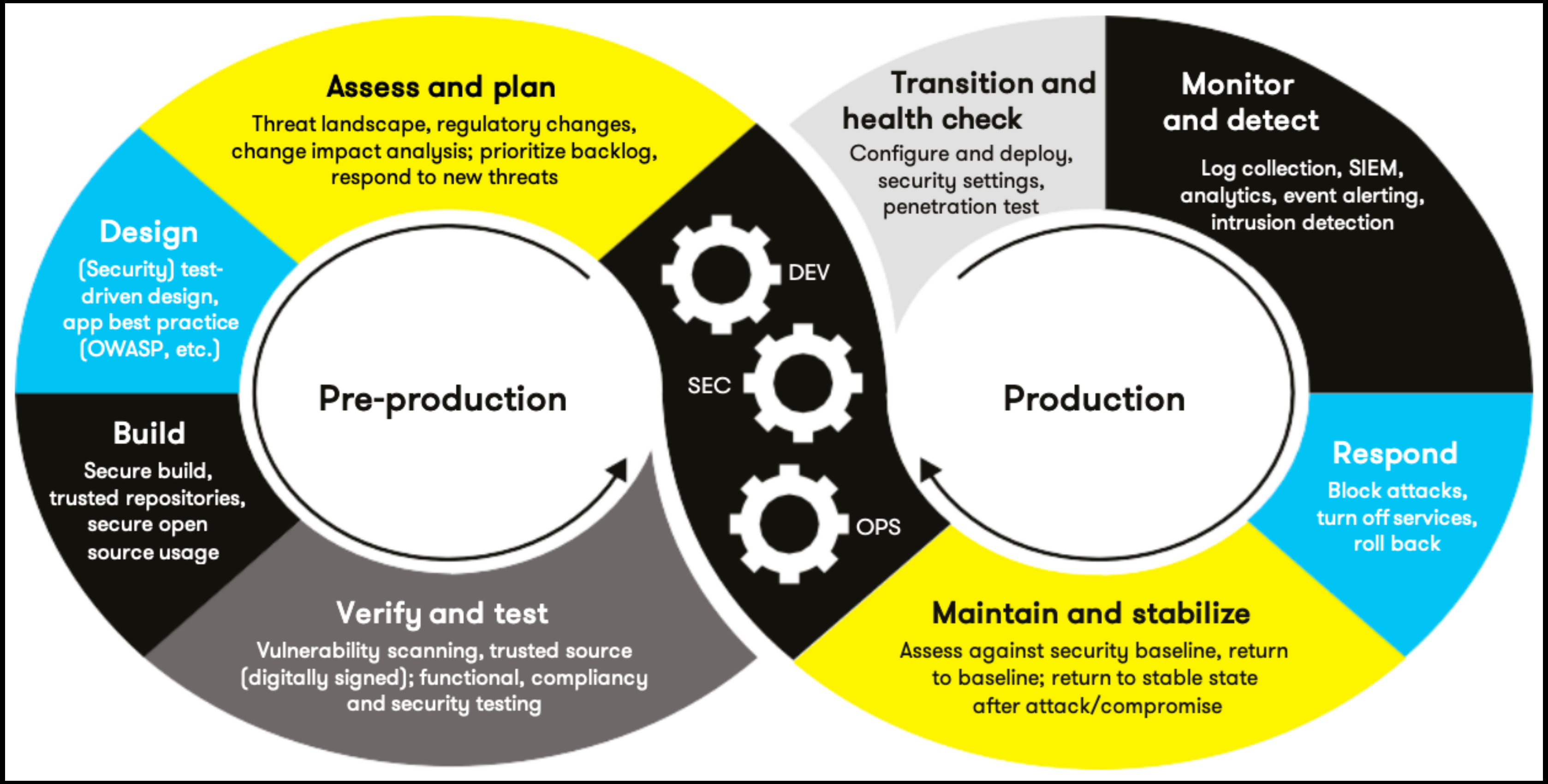
```
TEST_F(CollectionTest, NullOrNotTheCollection)
{
    add_entries(3);//set size
    collection->empty();//checking if collection is empty
    ASSERT_FALSE(collection == NULL);//verifying not null
    ASSERT_TRUE(collection != NULL);//verifying collection is valid

}
```

```
[ RUN      ] CollectionTest.NullOrNotTheCollection
[       OK ] CollectionTest.NullOrNotTheCollection (0 ms)
```

Green Pace

AUTOMATION SUMMARY



Green Pace

TOOLS

- The DevSecOps pipeline integrates security practices into the DevOps pipeline, where security becomes a shared responsibility of development, security, and operations teams. It emphasizes automation, continuous security testing, and compliance scanning across the software development lifecycle (SDLC).
- We utilized CPPCheck to check for vulnerabilities during the “Verify and Test phase. Cppcheck analyzes coding patterns, memory leaks, and best practices to improve code quality.

Green Pace

RISKS AND BENEFITS

Acting Now

- Benefits:
  - •Immediate problem resolution, preventing escalation.
  - •Competitive advantage through early action.
  - •Improved efficiency, security, or compliance.
- Risks:
  - •Immediate problem resolution, preventing escalation.
  - •Competitive advantage through early action.
  - •Improved efficiency, security, or compliance.

Waiting

- Benefits:
  - •More time for analysis and better decision-making.
  - •Reduced initial costs by avoiding rushed investments.
  - •Opportunity to learn from others' mistakes or advancements
- Risks:
  - •Problems may worsen over time, leading to higher costs later.
  - •Lost opportunities for innovation or competitive edge.
  - •Reduced stakeholder confidence due to inaction.

Green Pace

RECOMMENDATIONS

- Security Policies are updated and change without notice. Keeping an up-to-date version check of policies and procedures.
- Set standard list of Policies for every project and then document what kind of projects built and what policies are used for those types of projects. This can lead to having a standard list of polices to test against and then add as needed depending on the context of the project.

Green Pace

CONCLUSIONS

- Provide security training for all staff.
- Use up to date standards.
- Use checkers when able and collect data to share experiences with team.
- Run internal audits.

Green Pace

REFERENCES

- *SEI CERT C++ Coding Standard - SEI CERT C++ Coding Standard - Confluence*. (n.d.). Wiki.sei.cmu.edu. https://wiki.sei.cmu.edu/confluence/pages/viewpage.action?pageId=88046682

- *Cppcheck - A tool for static C/C++ code analysis*. (n.d.). Cppcheck.sourceforge.io. https://cppcheck.sourceforge.io/

Green Pace