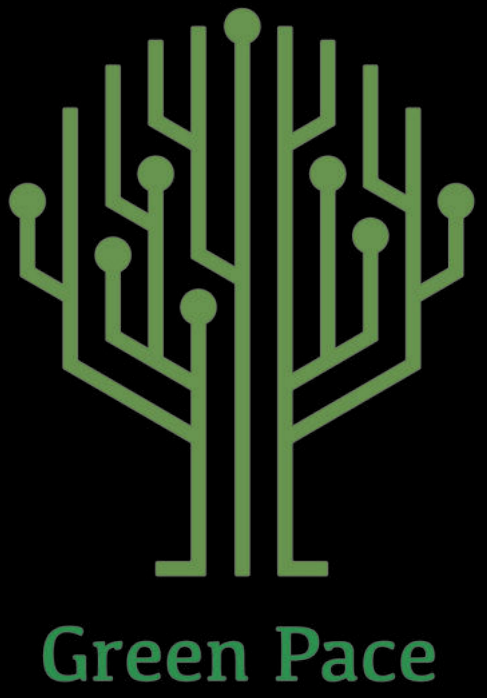
# Green Pace

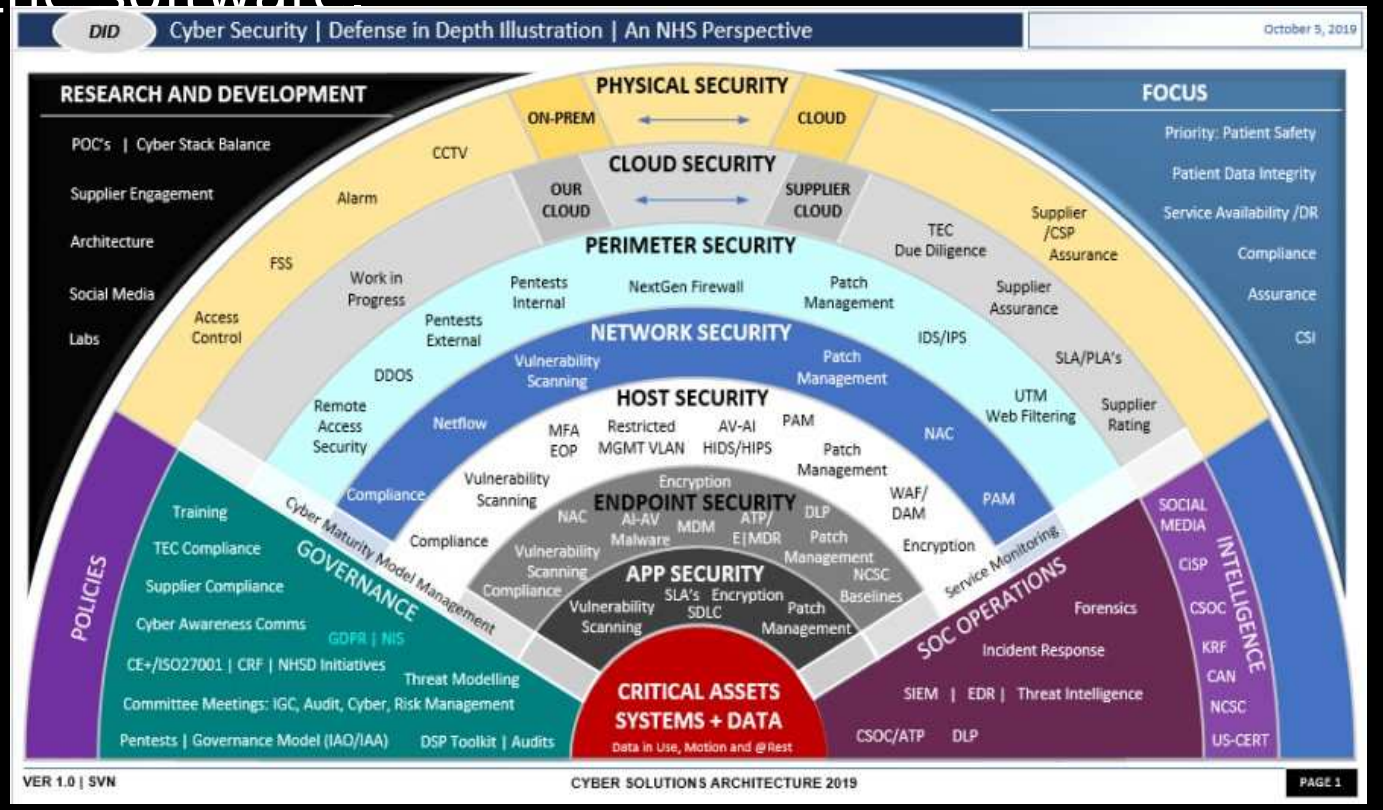
Security Policy Presentation

Developer: *Tabitha Tallent*




# OVERVIEW: DEFENSE IN DEPTH

This security policy utilizes defense in depth to define acceptable coding standards, in accordance with security best practices, for maintaining the safety of any data being handled, and the overall security of the software.



# THREATS MATRIX

**Likely**
Likely/probable to happen, but with a low/medium severity

**High Priority**
Likely to happen with a high severity

**Low Priority**
Unlikely to happen with a low severity

**Unlikely**
Unlikely to happen, but with a medium/high severity

| | |
|---|---|
| Likely<br>STD-008-CPP<br>STD-009-CPP | High Priority<br>STD-003-CPP<br>STD-004-CPP<br>STD-005-CPP<br>STD-010-CPP |
| Low priority<br>STD-002-CPP<br>STD-007-CPP | Unlikely<br>STD-001-CPP<br>STD-006-CPP |

# 10 PRINCIPLES

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

# CODING STANDARDS

1. STD-003-CPP – Guarantee that storage for strings has sufficient space for character data and the null terminator
2. STD-004-CPP – Exclude user input from format strings
3. STD-005-CPP – Do not access freed memory
4. STP-010-CPP – Do not dereference null pointers
5. STD-001-CPP – Ensure that operations on signed integers do not result in overflow
6. STD-008-CPP – Ensure your random number generator is properly seeded
7. STD-009-CPP – Free dynamically allocated memory when no longer needed
8. STD-006-CPP – Understand the termination behavior of assert() and abort()
9. STD-007-CPP – Handle all exceptions
10. STD-002-CPP – Do not create incompatible declarations of the same f object

# ENCRYPTION POLICIES

- Encryption at Rest
  - Any stored data should always be encrypted, even if infrequently accessed
- Encryption in Flight
  - Data being transferred from one place to another should be encrypted to prevent man in the middle attacks
- Encryption in Use
  - Any data in use by the system should remain encrypted for the entire usage policy, even as it updates

# TRIPLE-A POLICIES

- Authentication
  - Use this to verify that a user is who they say they are, often manifests as a set of login credentials
- Authorization
  - What is a user allowed to do, generally this follows the principle of least privilege
- Accounting
  - Implementing logging for any activities performed/requests made c identify an attack or even the attempt earlier in the event

# Unit Testing

## EraseEmptiesCollection

```
// Test to verify erase(begin,end) erases the collection

TEST_F(CollectionTest, EraseEmptiesCollection) {
    // Add entries
    add_entries(632);

    // Erase collection
    collection->erase(collection->begin(), collection->end());

    // Verify collection is now empty (has a size of 0)
    ASSERT_TRUE(collection->empty());
    ASSERT_EQ(collection->size(), 0);
}
```

Memory management and allocation. (Positive Test)

**Test Detail Summary**

CollectionTest.EraseEmptiesCollection<CollectionTest> [EraseEmptiesCollectior

Source: test.cpp line 199

Duration: < 1 ms

# Unit Testing

## ReserveIncreasesCapacityAndNotSize

```
// Test to verify reserve increases the capacity but not the size of the collection

TEST_F(CollectionTest, ReserveIncreasesCapacityAndNotSize) {
    // Declare intiial capacity
    size_t initial_capcity = collection->capacity();

    // Set collection reserve
    collection->reserve(46);

    // Verify collection capacity is now greater than initial capacity
    ASSERT_GT(collection->capacity(), initial_capcity);
    // Verify that collection size is still 0
    ASSERT_EQ(collection->size(), 0);
}
```

Memory safety
(Positive Test)

**Test Detail Summary**

CollectionTest.ReserveIncreasesCapacityAndNotSize<CollectionTest> [ReserveIncreasesCapacityAndNo

Source: test.cpp line 212

Duration: < 1 ms

# Unit Testing

## ResizingIncreasesCollection

```
// Test to verify resizing increases the collection
TEST_F(CollectionTest, ResizingIncreasesCollection) {
    // Add entries
    add_entries(4);

    // Declare initial collection size
    size_t initial_size = collection->size();

    // Resize collection
    collection->resize(7);

    // Verify current collection size
    ASSERT_EQ(collection->size(), 7);
    // Verify that the current collection size is greater than the initial collection size
    ASSERT_GT(collection->size(), initial_size);
}
```

Memory optimization and resource handling.
(Positive Test)

**Test Detail Summary**

CollectionTest.ResizingIncreasesCollection<CollectionTest> [ResizingIncreasesCo

Source: test.cpp line 140

Duration: < 1 ms

# Unit Testing

## AccessingRemovedElementTerminatesProcess

Bounds and access checking.
(Negative Test)

```
// Negative test to verify the process will terminate if access is attempted on a removed element

TEST_F(CollectionTest, AccessingRemovedElementTerminatesProcess) {
    add_entries(1);
    auto it = collection->begin();
    collection->erase(collection->begin());
    ASSERT_DEATH(*it, ".*");
}
```

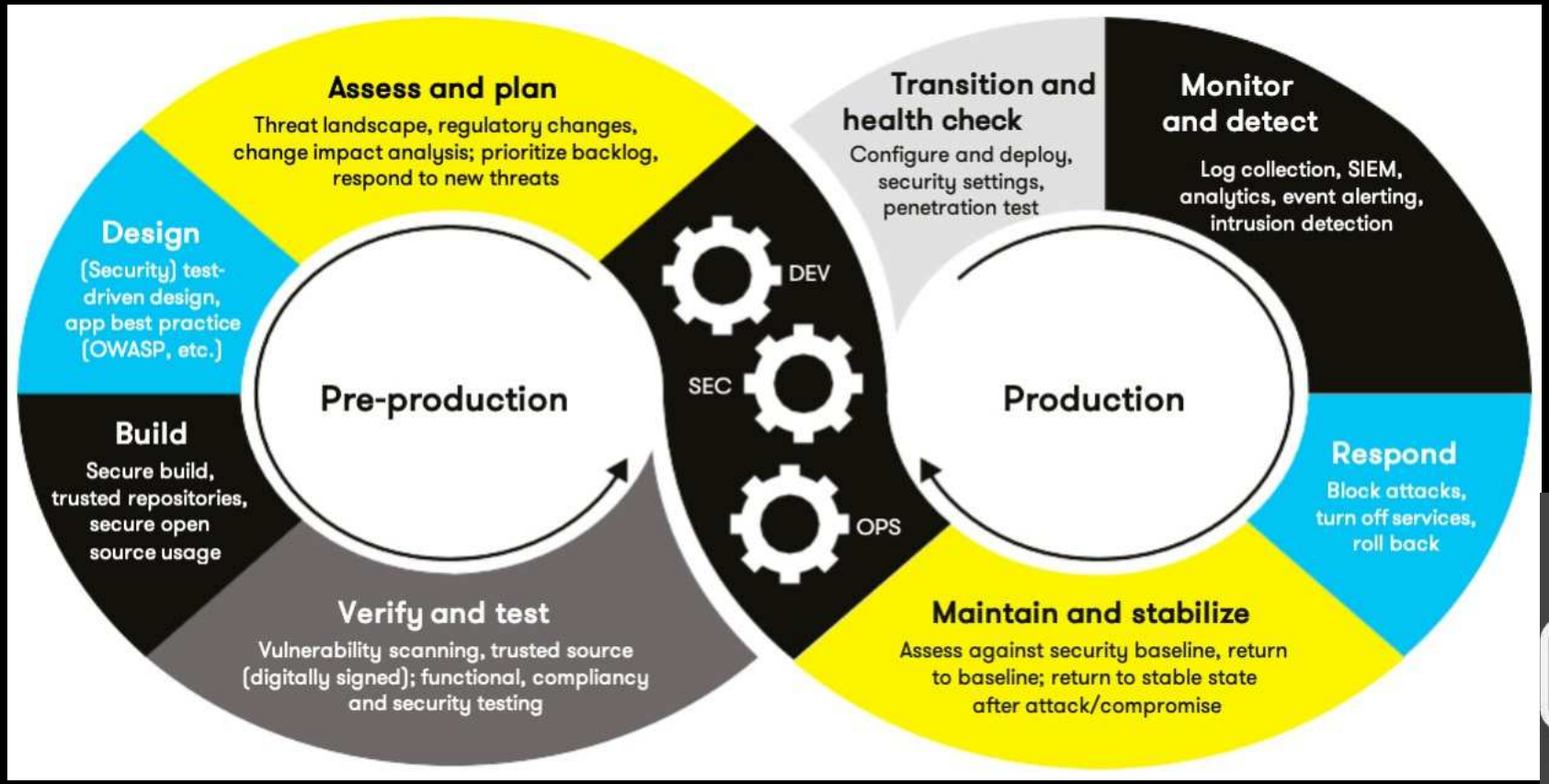
**Test Detail Summary**

CollectionTest.AccessingRemovedElementTerminatesProcess<CollectionTest> [AccessingRemovedElementTerminatesPr

Source: test.cpp line 248

Duration: 4.9 sec

# AUTOMATION SUMMARY



# TOOLS

- DevSecOps is a development life cycle that focuses on implementing security from the start of any project, all the way from the planning stage. Automation comes into play in several stages:
- Build
  - Static testing
- Verify and Test
  - Dynamic testing
- Transition and Health
  - Firewalls
- Monitor and Detect
  - Logging

# RISKS AND BENEFITS

- DO NOT LEAVE SECURITY UNTIL THE END
- Security is a critical aspect of any application, especially one handling user data, and it should be considered from the design and planning stages.
- Benefits of prioritizing security:
  - Easier to implement at the beginning
  - Better codebases
  - Increased coding skill for developers
- Risks of not prioritizing security:
  - Difficulty implementing security into existing codebase
  - Could result in a preventable security/data breach

(Don’t Leave Security For Last

# RECOMMENDATIONS

- Personnel training is important, people cannot follow a policy they are unfamiliar with.
- Maintain established, and utilized, repercussions for not following the security policy.
- Regularly review and update threats to the application based on new information and attack possibilities
- Remember to implement access control management and appropriate authentication protocols and authorization policies.
- Monitor and log everything that any user does on the system, automate this to alert for certain security concerns.
- Have a plan for handling a possible security concern for if it occurs.

# CONCLUSIONS

- Practice secure coding from the start
- Utilize the coding principles listed to help secure the application
- Follow all secure coding best practices, reference https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard regularly for updates and other relevant practices
- Ensure training is provided for all personnel on their expected cooperation with this security policy, identify and enforce consequences for failing to meet security expectations
- Practice security automation and tool utilization in accordance with the DevSec pipeline

# REFERENCES


- Don’t Leave Security for Last | SoftwareTestPro. (n.d.). https://www.softwaretestpro.com/dont-leave-security-for-last/
- Person, D. (2023, September 1). 15 Security Best practices for companies. Forbes. https://www.forbes.com/councils/forbestechcouncil/2025/02/21/unlocking-ais-potential-overcoming-barriers-to-adoption/
- SEI CERT C Coding Standard - SEI CERT C Coding Standard - Confluence. (n.d.). https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard


Green Pace