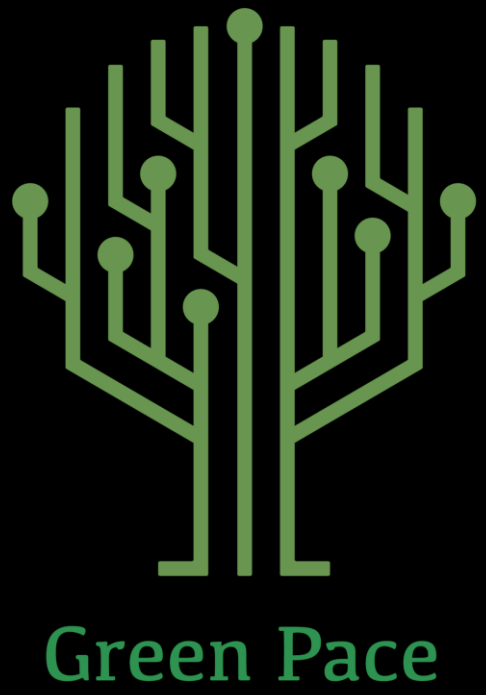
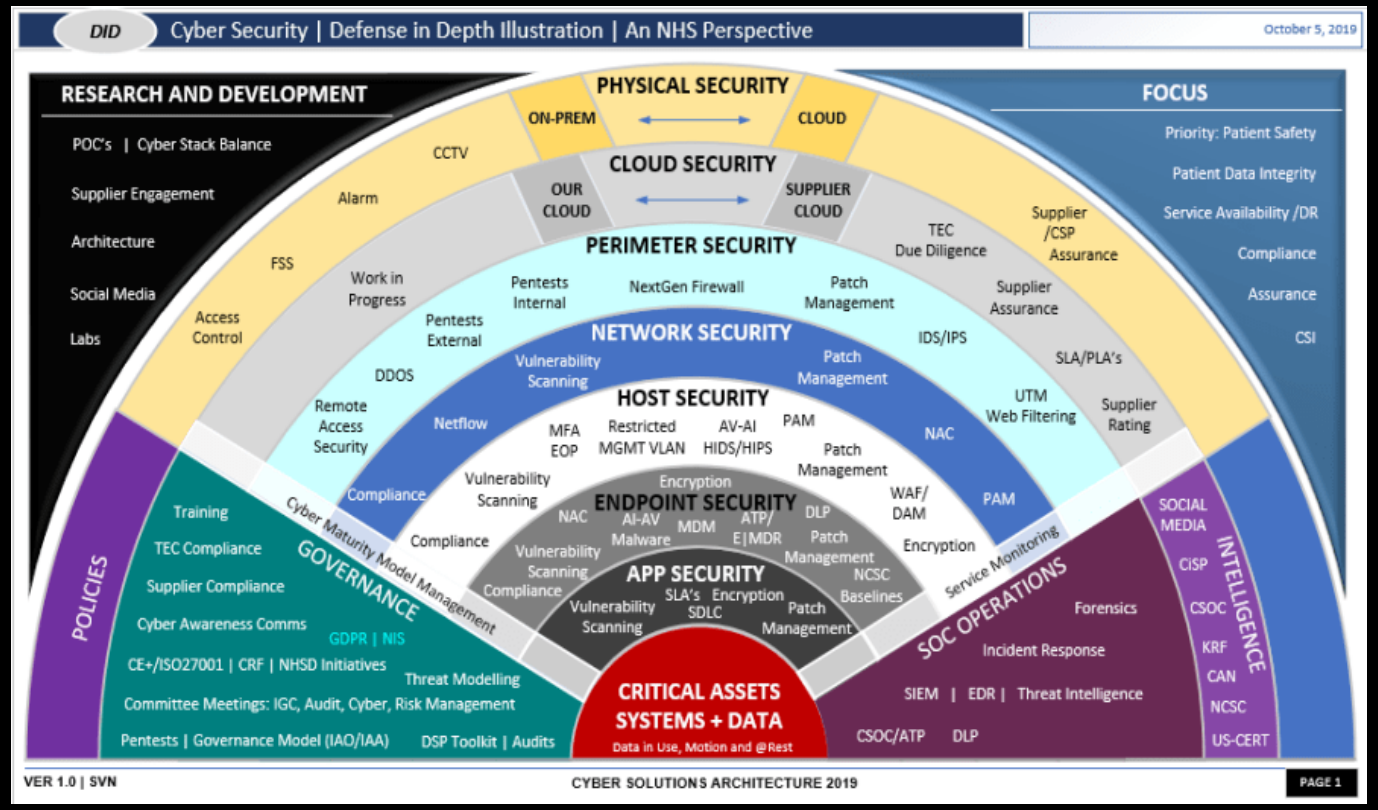
Green Pace

Security Policy Presentation

Developer: *Bryce Cooperrider*



OVERVIEW: DEFENSE IN DEPTH



THREATS MATRIX

Likely STR50-CPP (Data Type) INT50-CPP (Data Value) MSC11-C (Assertions) STR30-C (SQL Injection)	**Priority** STR50-CPP (Data Type) MEM31-C (Memory Protection) MSC11-C (Assertions) OOP53-C (OOP) STR30-C (SQL Injection)
Low priority FIO51-CPP (Input Output) ERR51-CPP (Exceptions)	**Unlikely** STR51-CPP (String Corre MEM31-C (Memory Prote ERR62-CPP (Error Hand

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. STR50-CPP: Guarantee that storage for strings has sufficient space for character data and the null terminator
- 2. INT50-CPP: Do not cast to an out-of-range enumeration value
- 3. STR51-CPP: Do not attempt to create a std::string from a null pointer
- 4. STR30-C: Do not attempt to modify string literals
- 5. MEM31-C: Free dynamically allocated memory when no longer needed
- 6. MSC11-C: Incorporate diagnostic tests using assertions
- 7. ERR51-CPP: Handle all exceptions
- 8. OOP53-CPP: Write constructor member initializers in the canonical order
- 9. FIO51-CPP: Close files when they are no longer needed
- 10. ERR62-CPP: Detect errors when converting a string to a number

ENCRYPTION POLICIES

- Encryption at rest
- Encryption in flight
- Encryption in use

TRIPLE-A POLICIES

- Authentication
- Authorization
- Accounting

Unit Testing

The following slides are examples of unit tests conducted

Is the max size of the collection greater than or equal to its size?

Asserting that the max size of the collection is greater than or equal to its current size leaves room for expansion

This can be taken a step further by testing the negative, assuring the collection is not less than or equal to its size

```
TEST_F(CollectionTest, IsMaxSizeGreaterThanOrEqualSize) {

    //Check for 0
    //is the collection empty?
    EXPECT_TRUE(collection->empty());
    //if empty, the size must be 0
    EXPECT_EQ(collection->size(), 0);
    //Verify max size is greater than or equal to size
    ASSERT_GE(collection->max_size(), collection->size());

    //Check for 1
    add_entries(1);
    //is the collection empty?
    EXPECT_FALSE(collection->empty());
    //if empty, the size must be 0
    EXPECT_EQ(collection->size(), 1);
    //Verify max size is greater than or equal to size
    ASSERT_GE(collection->max_size(), collection->size());

    //Check for 5
    add_entries(4);
    //is the collection empty?
    EXPECT_FALSE(collection->empty());
    //if empty, the size must be 0
    EXPECT_EQ(collection->size(), 5);
    //Verify max size is greater than or equal to size
    ASSERT_GE(collection->max_size(), collection->size());

    //check for 10
    add_entries(5);
    //is the collection empty?
    EXPECT_FALSE(collection->empty());
    //if empty, the size must be 0
    EXPECT_EQ(collection->size(), 10);
    //Verify max size is greater than or equal to size
    ASSERT_GE(collection->max_size(), collection->size());
```

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

Can the collection be decreased to zero?

Decreasing the collection size to zero effectively removes data from the collection

A step further would be to test the memory addresses to ensure no data is stored in them

```
TEST_F(CollectionTest, VerifyDecreaseCollectionToZero) {

    //add 5 entries
    add_entries(5);
    //is the collection empty?
    EXPECT_FALSE(collection->empty());
    //If empty, size must be 0
    EXPECT_EQ(collection->size(), 5);
    //resize collection to 0
    collection->resize(0);
    //is the collection still empty?
    EXPECT_TRUE(collection->empty());
    //If not empty, what must the size be?
    ASSERT_EQ(collection->size(), 0);
}
```

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

Does resizing the collection increase its size?

Resizing the collection is needed to add additional information to the collection

Taking it a step further would be trying to write information to the reserved space

```
TEST_F(CollectionTest, VerifyRezisingIncreasesCollection) {

    //is the collection empty?
    EXPECT_TRUE(collection->empty());
    //If empty, size must be 0
    EXPECT_EQ(collection->size(), 0);
    //resize collection to 5
    collection->resize(5);
    //is the collection still empty?
    EXPECT_FALSE(collection->empty());
    //If not empty, what must the size be?
    ASSERT_EQ(collection->size(), 5);
}
```

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

Does shrinking the collection to fit work as intended?

It is often necessary to remove space from the collection. Using shrink to fit is a great way to free up memory space when not needed

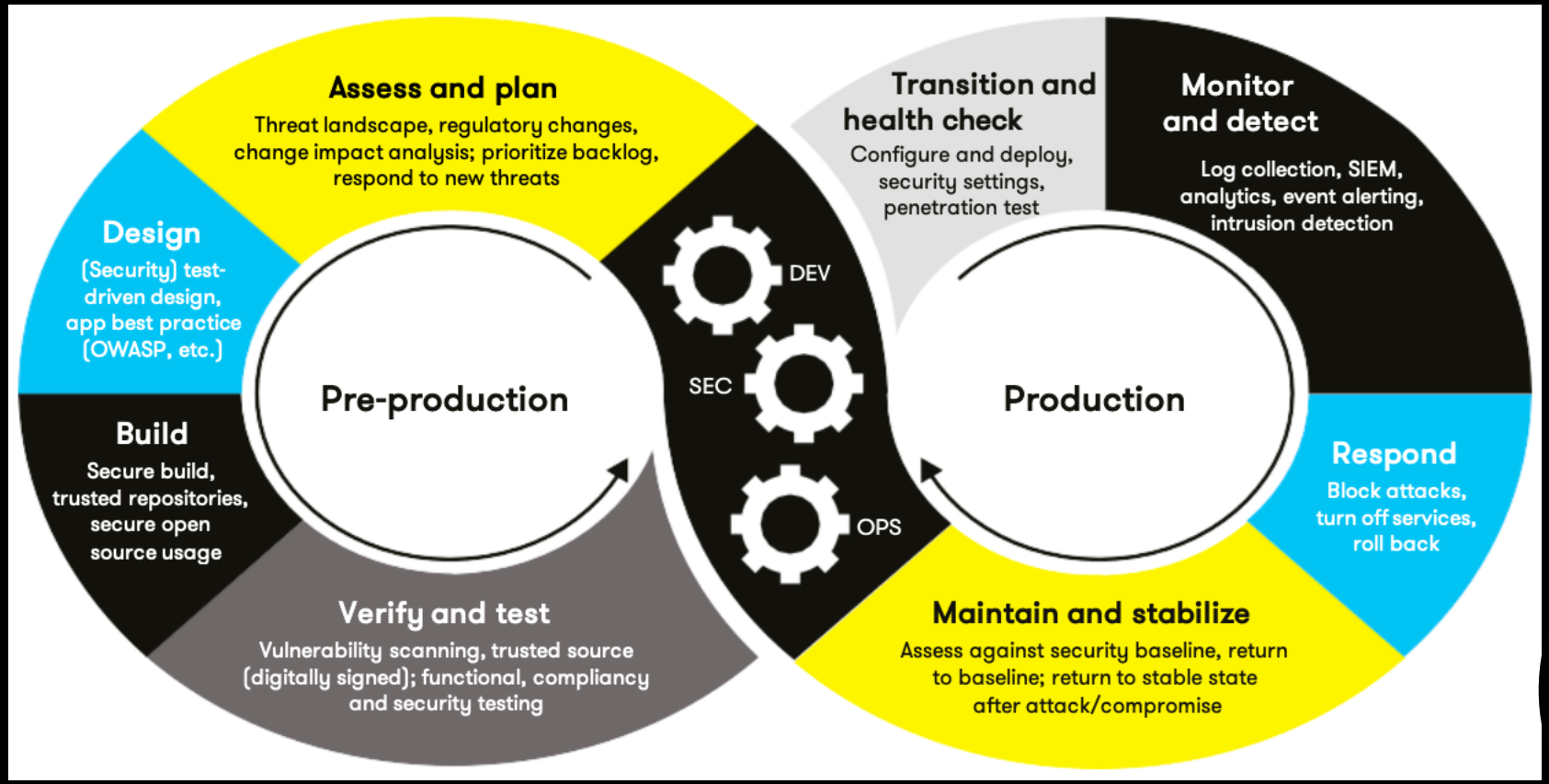
Further testing could include resizing to different amounts, including resizing to zero

```
TEST_F(CollectionTest, VerifyShrinkToFit) {

    //add 5 entries
    add_entries(5);
    //is the collection empty?
    EXPECT_FALSE(collection->empty());
    //if not empty, what must the size be?
    EXPECT_EQ(collection->size(), 5);
    //Set capacity to 10
    collection->capacity(), 10;
    //confirm capacity is greater than collection length
    EXPECT_GT(collection->capacity(), collection->size());
    //resize capacity to equal length
    collection->shrink_to_fit();
    //check if capacity equals length
    ASSERT_EQ(collection->capacity(), collection->size());
}
```

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

AUTOMATION SUMMARY



RISKS AND BENEFITS

- Benefits of acting now:
  - Acting in the moment can mitigate expected attacks.
  - Decreased long term costs.
  - Stronger initial security
- Risks of acting now:
  - Could be protecting against the wrong attack.
  - Could end up with misused resources.
- Benefits of acting later:
  - We know exactly what we need to defend against.
- Risks of acting later:
  - Could be costly to the company.
  - Could increase costs due to development time.
  - Risk of damaging company reputation.

RECOMMENDATIONS

- Implement Security Automation
- Invest in Staff Training
- Conduct a Risk Assessment On the Current Policies
- Enable Cross Team Communication

CONCLUSIONS

- Security should be thought of before, during, and after development. It is an ongoing battle that needs constant attention to mitigate any potential risks and is always evolving.
- Being proactive with security is the best-case scenario to mitigate any reputation and data loss for the company.

REFERENCES

- [Provide APA-style references with links to resources, articles, and videos that you used in your presentation.]

Green Pace