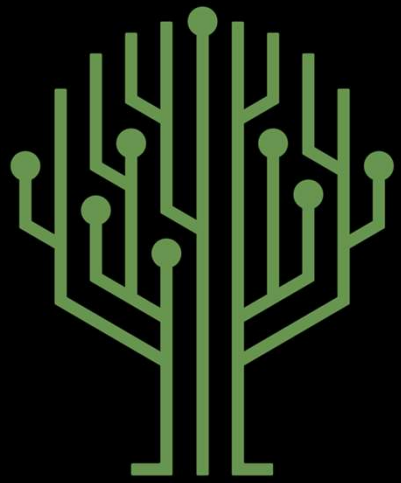
Green Pace

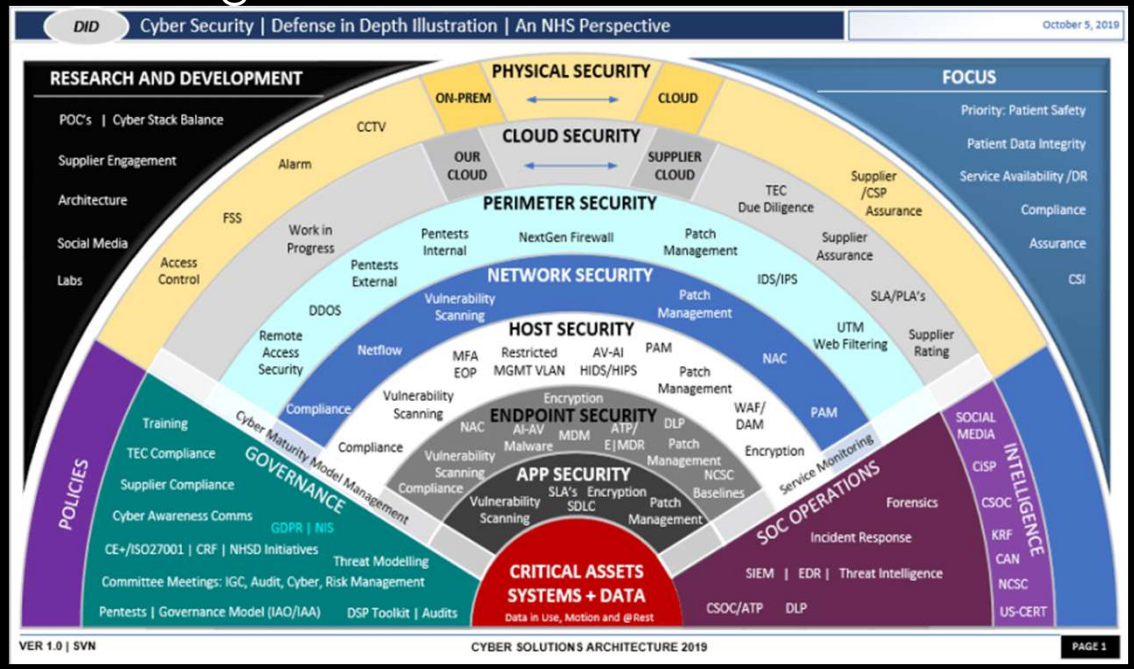
Security Policy Presentation

Developer: *Richard Schall*



OVERVIEW: DEFENSE IN DEPTH

This is the new security policy. This policy is needed to ensure all developers are coding to the same set of standards for security.



THREATS MATRIX

	Low	Medium	High
Likely	L2: Medium severity, probable to happen	L1: High severity, likely to happen	L1:High severity, likely to happen
Probable	L2: Medium severity, probable to happen	L1: High severity, likely to happen	L1:High severity, likely to happen
Unlikely	L3: Low Severity, unlikely to happen	L2: Medium severity, probable to happen	L2: Medium severity, probable to happen

10 PRINCIPLES

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

CODING STANDARDS

- [STD-001-CPP]: Do not use floating-point variables as loop counters.
- [STD-002-CPP]: Ensure that unsigned integer operations do produce overflow and underflow.
- [STD-003-CPP]: Make sure the storage for strings has enough space for character data and null terminator.
- [STD-004-CPP]: Prevent SQL injection.
- [STD-005-CPP]: Do not read uninitialized memory.
- [STD-006-CPP]: Use a static assertion to test the value of a constant expression.
- [STD-007-CPP]: Handle all exceptions.
- [STD-008-CPP]: Range check element access.
- [STD-009-CPP]: Close files when they are no longer needed.
- [STD-010-CPP]: Value-returning functions must return a value from all exit paths.

ENCRYPTION POLICIES

- Encryption in rest: Encryption at rest refers to data that sits statically in tables. All data and encryption keys will be stored separately for each other. All sensitive data must be encrypted like passwords and any other data classified as sensitive. (Cairns & Somerfield, 2017)
- Encryption at flight: Attackers can retrieve data as it is transmitted to a client, these are referred to as man-in-the-middle attacks. Sensitive data should not be transmitted in plain text, it should be encrypted. (Cairns & Somerfield, 2017)
- Encryption in use: An example of encryption in use is password verification and using a hash of the original password data to do comparisons to what the user enters. (Cairns & Somerfield, 2017)

TRIPLE-A POLICIES

- Authentication: Authentication covers identifying a user and verifying who they are. Authentication ensures that only approved users gain access to a network and computer system. This is crucial to security. (O'Carroll, 2018)
- Authorization: After a user is authenticated, authorization determines what parts of the computer program, system, or network the user is permitted to access. Authentication also controls what actions users are permitted to take. A user should never be given more access than they need. (O'Carroll, 2018)
- Accounting: Accounting tracks the activity of users, what activity occurred when and by who in the form of a log. In the event of an attack or security event, having proper accounting will help system administrators perform a post-mortem on the security event to analyze what happened. This will determine what counter-measure activity needs to occur. (O'Carroll, 2018)

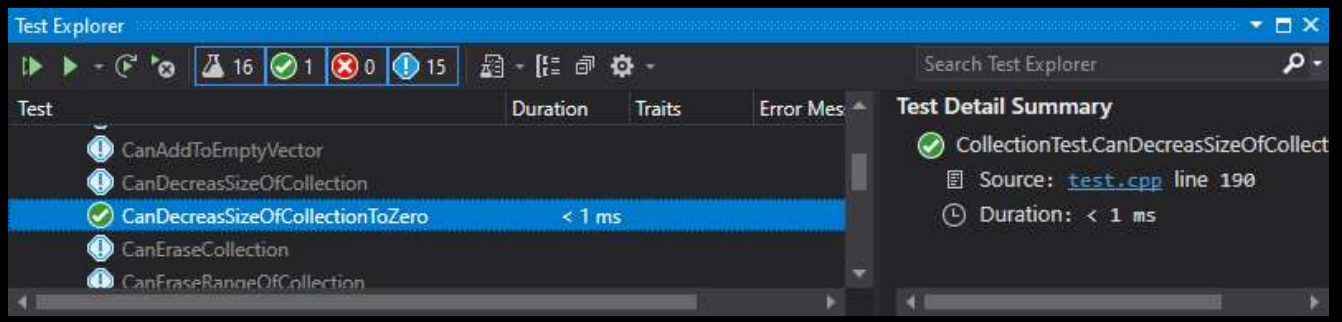
Unit Testing

Does Re-Sizing a Collection to Zero Decrease a Collection to Zero (positive)?

Code:

```
// TODO-07: Create a test to verify resizing decreases the collection to zero
TEST_F(CollectionTest, CanDecreasSizeOfCollectionToZero)
{
    // Add three entries to the collection -Richard
    add_entries(3);

    // Verify the size of the collection is three
    ASSERT_EQ(collection->size(), 3);

    // Resize the collection to zero and verify size is zero
    collection->resize(0);

    ASSERT_EQ(collection->size(), 0);
}
```

Results:



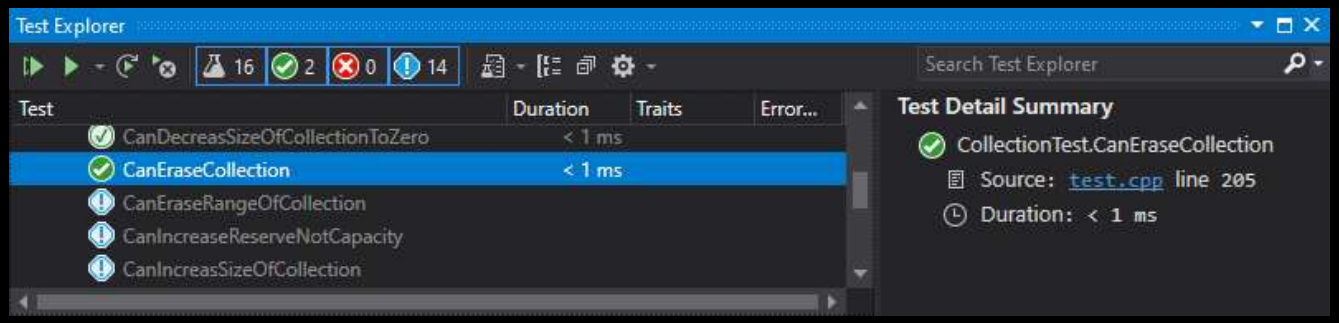
Unit Testing

Does Calling Clear on the Collection Erase the Collection (positive)?

Code:

```
// TODO-08: Create a test to verify clear erases the collection
TEST_F(CollectionTest, CanEraseCollection)
{
    // Add three entries to the collection -Richard
    add_entries(3);

    // Verify the size of the collection is three
    ASSERT_EQ(collection->size(), 3);

    // Erase the collection, verify size is zero, verify collection is zero - Richard
    collection->clear();

    EXPECT_EQ(collection->size(), 0);

    // Assert true if the collection is empty - Richard
    ASSERT_TRUE(collection->empty());
}
```

Results:



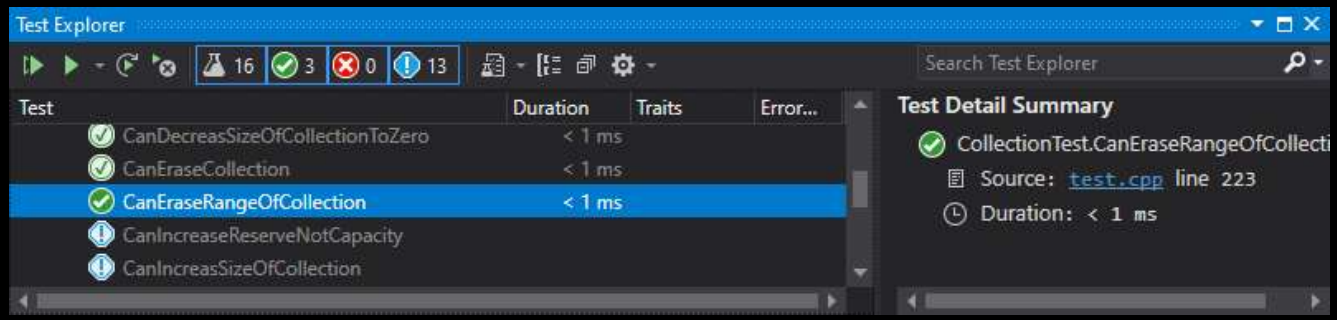
Unit Testing

Does Calling Erase (begin, end) erase the collection (positive)?

Code:

```
// TODO-09: Create a test to verify erase(begin,end) erases the collection
TEST_F(CollectionTest, CanEraseRangeOfCollection)
{
    // Add three entries to the collection -Richard
    add_entries(12);

    // Verify the size of the collection is three
    ASSERT_EQ(collection->size(), 12);

    // Erase the collection, verify size is zero, verify collection is zero - Richard
    collection->erase(collection->begin(), collection->begin() + 12);

    EXPECT_EQ(collection->size(), 0);

    // Assert true if the collection is empty - Richard
    ASSERT_TRUE(collection->empty());
}
```

Results:



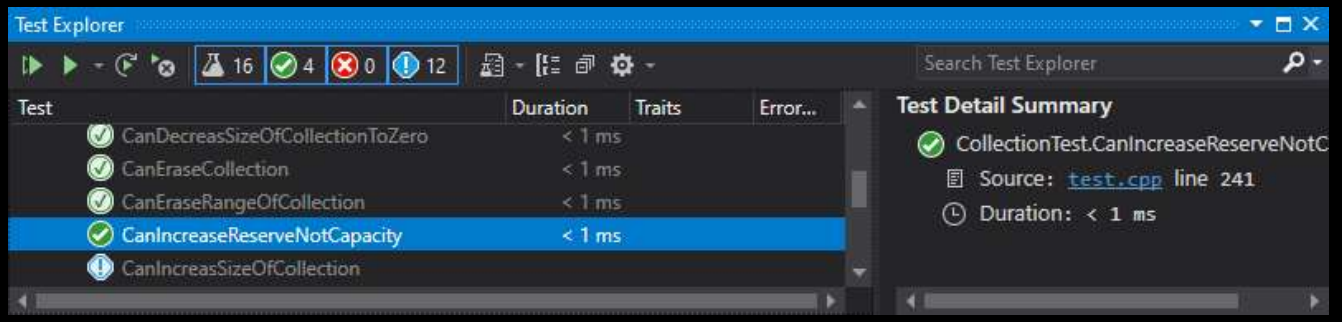
Unit Testing

Does Calling Reserve increase the capacity but not the size (positive)?

Code:

```
// TODO-10: Create a test to verify reserve increases the capacity but not the size of the collection
TEST_F(CollectionTest, CanIncreaseReserveNotCapacity)
{
    // Add three entries to a collection
    add_entries(3);

    // Check the size of the collection equals three
    ASSERT_EQ(collection->size(), 3);

    // Use reserve to increase the capacity by 2, confirm capacity equals five
    collection->reserve(5);

    ASSERT_EQ(collection->capacity(), 5);

    // Confirm size is three
    ASSERT_EQ(collection->size(), 3);
}
```

Results:



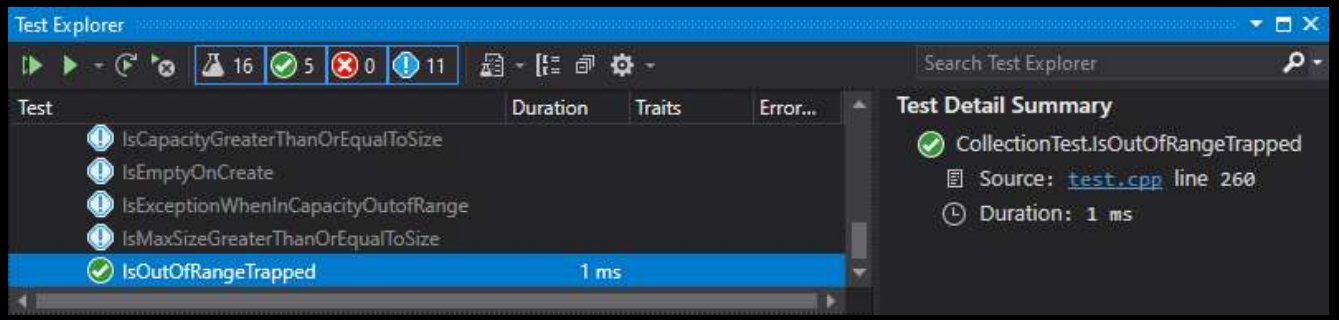
Unit Testing

Does Calling At() for an index out of bounds throw an exception (negative)?

Code:

```
// TODO-11: Create a test to verify the std::out_of_range exception is thrown when calling at() with an index out of bounds
// NOTE: This is a negative test
TEST_F(CollectionTest, IsOutOfRangeTrapped)
{
    try
    {
        // Add three entries to a collection
        add_entries(3);

        // Check the size of the collection equals three, reference an element out of range, verify exception
        ASSERT_EQ(collection->size(), 3);

        std::cout << collection->at(5);
    }
    catch (std::exception& e)
    {
        bool caught = true;

        ASSERT_TRUE(caught);
    }
}
```

Results:



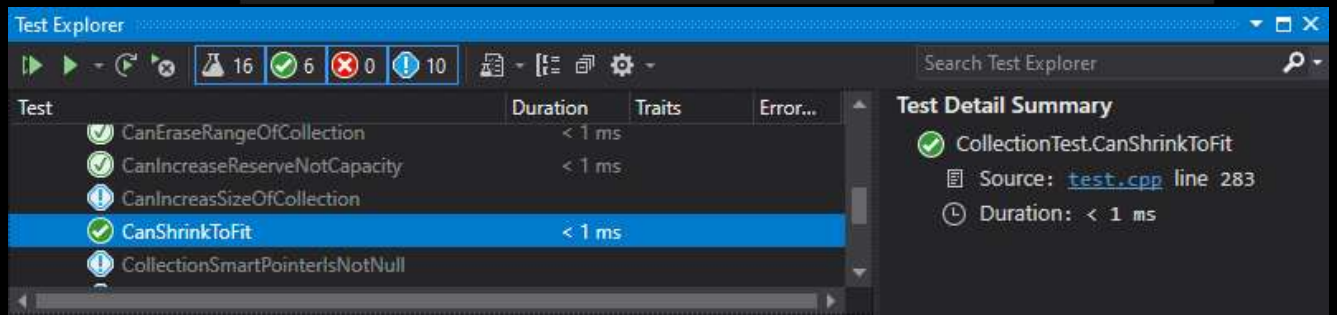
Unit Testing

Does Lowering Size and Calling Shrink Change the Capacity (positive)?

Code:

```
// TODO-12: Create 2 unit tests of your own to test something on the collection - do 1 positive & 1 negative
// Verify Shrink-To-fit reduces capcity to size
TEST_F(CollectionTest, CanShrinkToFit)
{
    // Add three entries to a collection
    add_entries(3);

    // Check the size of the collection equals three
    ASSERT_EQ(collection->size(), 3);

    // Use reserve to increase the capacity by 2, confirm capacity equals five
    collection->reserve(5);

    ASSERT_EQ(collection->capacity(), 5);

    // Confirm size is three
    ASSERT_EQ(collection->size(), 3);

    // Shrink the capacity verify now is 3
    collection->shrink_to_fit();

    ASSERT_EQ(collection->capacity(), 3);
}
```

Results:



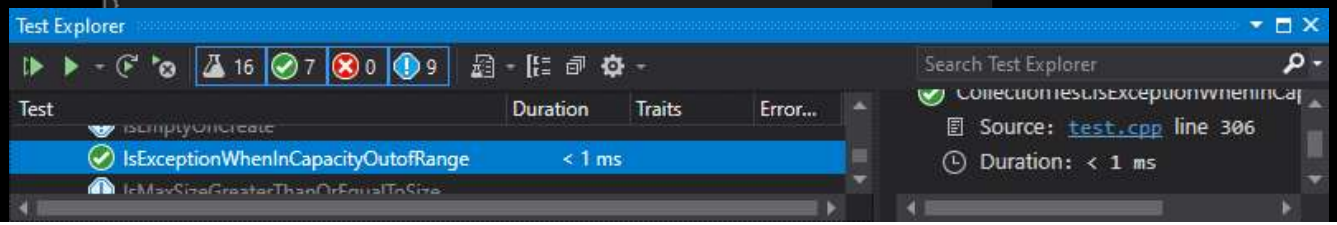
Unit Testing

Does Re-Sizing a Collection to Zero Decrease a Collection to Zero (positive)?

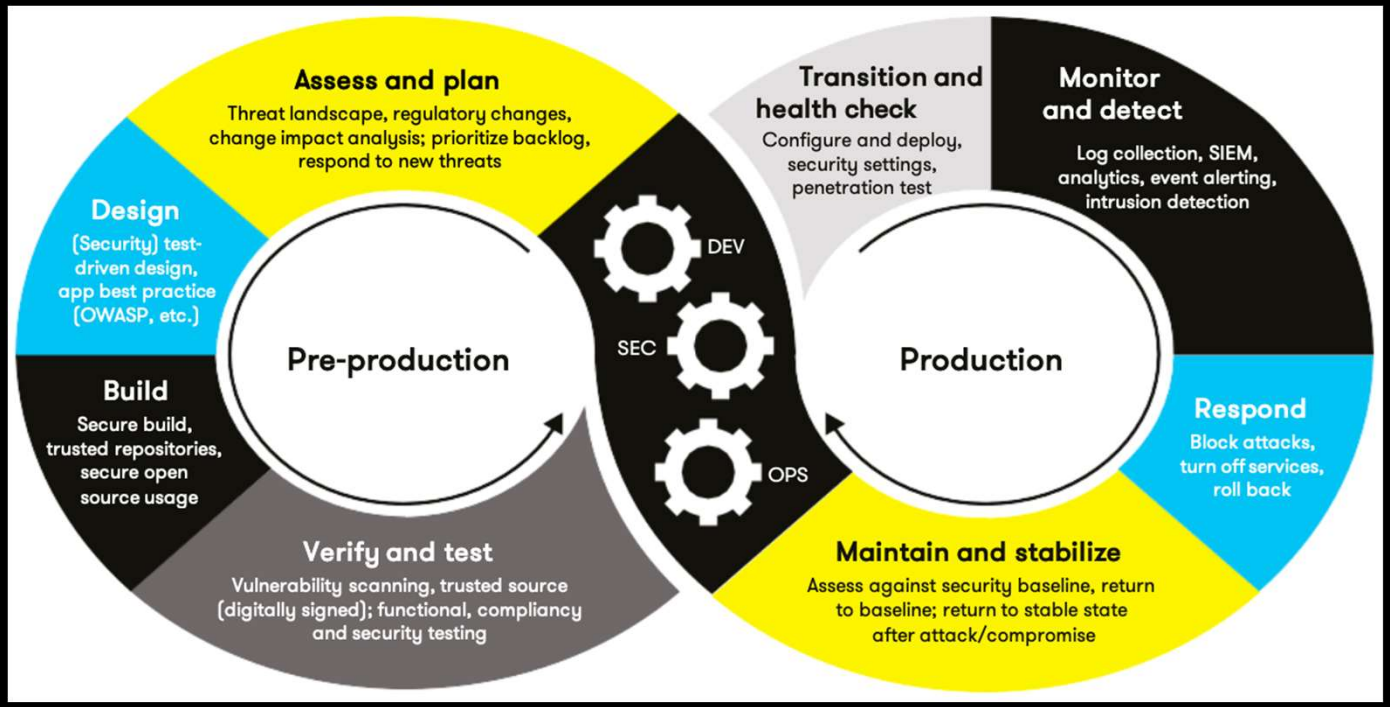
Code:

```
// TODO-13: Verify referencing an entry in a increased capacity produces an exception - NEGATIVE TEST
TEST_F(CollectionTest, IsExceptionWhenInCapacityOutofRange)
{
    try
    {
        // Add three entries to a collection
        add_entries(3);

        // Check the size of the collection equals three
        ASSERT_EQ(collection->size(), 3);

        // Use reserve to increase the capacity by 2, confirm capacity equals five
        collection->reserve(5);

        ASSERT_EQ(collection->capacity(), 5);

        // Confirm size is three
        ASSERT_EQ(collection->size(), 3);

        std::cout << collection->at(4) << std::endl;
    }
    catch (std::exception& e)
    {
        bool caught = true;

        ASSERT_TRUE(caught);
    }
}
```

Results:



AUTOMATION SUMMARY



TOOLS

- DevSecOps will occur the same frequency as DevOps. For a waterfall software development life cycle, this activity may occur at the end of development. In an iterative software development lifecycle should occur at the end of every iteration and for every commit if the security features are a part of a unit testing program. Automatic security testing should only be running on code that poses a risk, not on entire project.
- The additions to DevOps to support DevSecOps will be unit test written specifically to test for security vulnerability. These tests will run at every commit, nightly, and at a minimum at the end of an iteration or sprint. This is on the development side of DevSecOps. The rules stated above will need one of the forms of automatic static testing broken out in the automation table of each rule.
- On the operations side DevOps will need to include integrity checks and defense-in-depth measurements for prevention. For detection, network monitoring and penetration tests will need to be implemented.
- Tools:
  - Astree': This is static analyzer that can be used to run static tests on code and will be used to build automated tests as a part for DevSecOps. (Webmaster@absint.com, 2021)
  - CodeSonar: This is another static analyzer that can be used to run static tests on code and will be used to build automated tests as a part for DevSecOps. (Rules explorer 2021)

RISKS AND BENEFITS

- Problems and Risks
  - Adherence: How does Green Pace adhere to the new security policy?
  - Time: What will be the impact to development schedules when adopting the new security policy?
  - Resources: Will additional resources need to be made available to adhere and administer the security policy?
- Solutions
  - Proper training of associates will facilitate the adherence to the security policy.
  - Make adhering to the new security policy a part of the project management of all future projects. Build in time to adhere to the security policy.
  - Add additional resources for code reviews on the adherence to the security policy.
- Benefits
  - Adhering to the security policy will improve quality and result in less errors in released applications.
  - The security policy could help Green Space avoid future attacks that can result in capital losses for the company.
- Steps
  - Train associates on the security policy
  - Introduce the security policy as part of the software development life cycle
    - Make the security policy a part of the project plan
    - Make the security policy a part of code reviews.

RECOMMENDATIONS

- The gaps in the policy include the unknown. Green Pace needs to constantly analyze threats and update the security policy as needed.
  - Keep current with risks impacting the industry.
  - Update the security policy to reflect current risks
- The security policy should be viewed as a living document and be reviewed and maintained frequently.
- Put in place a training program for the new security policy.
- Have a system of auditing code for adherence to policy. Make the security policy a part of the code review process.

CONCLUSIONS

- The creation of the security policy is the first step in making sure Green Pace has a long and prosperous future.
- This is only the beginning of the security policy. The policy needs to be a living document that reflects the current risks impacting the industry.
- Security policy only works if it is followed, the associates of Green Pace will need to come together as a team to make the policy a success.

REFERENCES

- **Cairns, C. & Somerfield, D. (2017, January 05)**. The basics of web application security. Retrieved April 10, 2021, from https://martinfowler.com/articles/web-security-basics.html
- **CERT. (2020).** Confluence. Retrieved March 19, 2021, from https://wiki.sei.cmu.edu/confluence/display/c/SEI+CERT+C+Coding+Standard
- **O'Carroll, B. (2018, November 27)**. What is AAA Security? An introduction to authentication, authorisation and accounting. Retrieved April 10, 2021, from https://codebots.com/application-security/aaa-security-an-introduction-to-authentication-authorisation-accounting
- **Rules explorer. (n.d.)**. Retrieved April 16, 2021, from https://rules.sonarsource.com/
- **Webmaster@absint.com. (n.d.)**. Fast and sound static analysis. Retrieved April 16, 2021, from https://www.absint.com/astree/index.htm

Green Pace