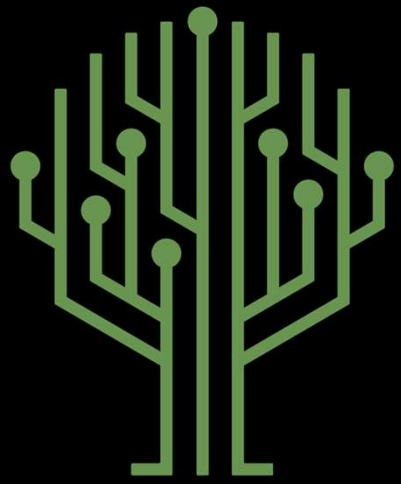
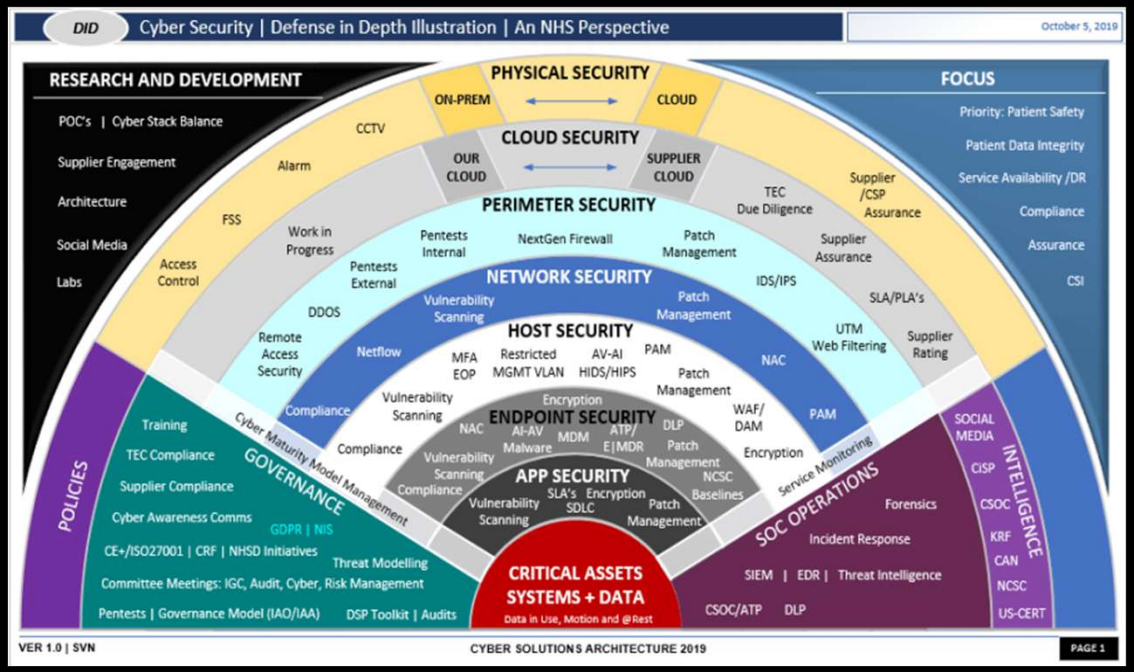
# Green Pace

Security Policy Presentation

Developer: Matt Jackson
Southern New Hampshire University




# OVERVIEW: DEFENSE IN DEPTH




Green Pace

# THREATS MATRIX

- eCrime accounts for most intrusions
- Most intrusions use social engineering
- Hactivism unlikely to threaten customer data
- Sabotage is unlikely but has a high impact

| | |
|---|---|
| Likely<br>eCrime | Priority<br>Social<br>Engineering |
| Low priority<br>Hacktivism | Unlikely<br>Sabotage |

# 10 PRINCIPLES

| Principles | Standards |
|---|---|
| Adopt a Secure Coding Standard | |
| Practice Defense in Depth | |
| Use Effective Quality Assurance Techniques | |
| Architect and Design for Security Policies | • Do not use C standard free() to deallocate objects allocated with new<br>• Handle all exceptions<br>• Make declarations unambiguous |
| Adhere to the Principle of Least Privilege | • Parameterize user input using prepared statements |
| Default Deny | • Parameterize user input using prepared statements |
| Validate Input Data | • Ensure data values do not wrap around<br>• Ensure access values are within the range of the string length<br>• Parameterize user input using prepared statements<br>• Do not use assertions to validate input |
| Sanitize Data Sent to Other Systems | • Close files when they are not needed. |
| Heed Compiler Warnings | • Use explicit, C++ casting conventions when converting data types<br>• Do not use C standard free() to deallocate objects allocated with new<br>• Handle all exceptions |
| Keep it Simple | • Make declarations unambiguous<br>• Do not assume order of evaluation will ensure side effects are executed in that order |

# CODING STANDARDS

- Parameterize user input using prepared statements
- Ensure access values are within the range of the string length
- Do not use C standard free() to deallocate objects allocated with new
- Close files when they are not needed.
- Handle all exceptions
- Do not use assertions to validate input
- Make declarations unambiguous
- Ensure data values do not wrap around
- Use explicit, C++ casting conventions when converting data types
- Do not assume order of evaluation will ensure side effects are executed in that order

# ENCRYPTION POLICIES

| Type | Policy |
| --- | --- |
| Encryption at rest | Data that is not currently being used or transmitted is considered “at rest”. Because data could be at rest for a prolonged period of time, a means of encryption that is not susceptible to brute force is best. For mobile devices, full-disk encryption is recommended in case the device is lost or stolen. |
| Encryption in flight | In flight data is moving from one location to another, be it through an email, file transfer protocol, web upload/download or the like. The best way to secure data in transit is through shared key encryption, most often asymmetrical, which solves the key distribution problem. |
| Encryption in use | When data is opened by an application or user, it is considered in use. Data in use can be protected by access controls and permissions. |

# TRIPLE-A POLICIES

| Triple-A Framework | Policy |
|---|---|
| Authentication | • Identifying a user<br>• Login with username / password<br>• MFA |
| Authorization | • Tasks or data a user can access<br>• Once authenticated, their authorization policy is applied.<br>Database changes<br>• Processes and files accessed |
| Accounting | • Log information about a user's session<br>• Verify policies are properly applied<br>• Analysis of resource usage<br>• Log user creation<br>Ensure policies can be checked for compliance and enforcement. |

# Unit Testing

- Add five values to a collection  (Positive test)
- Verify resize decrease  (Positive test)
- Throw out-of-range exception (Negative test)
- Search for removed value (Negative test)

# Add Five Values to Collection

```
// TODO: Create a test to verify adding five values to collection
TEST_F(CollectionTest, CanAddFiveValuesToVector)
{
    add_entries(rand() % 100);       // establish non-empty colleciton of up to 100 entries
    size_t size = collection->size();
    add_entries(5);
    EXPECT_EQ(collection->size(), size + 5);
}
```

# Verify resize decrease

```
// TODO: Create a test to verify resizing decreases the collection
TEST_F(CollectionTest, ResizeDecrease) {

    add_entries(rand() % 100 + 1 );             // ensure collection has random amount of entries in it
    size_t size = collection->size();

    size_t dec = collection->max_size();
    while (dec >= collection->size()) {
        dec = rand() % 100 + 1;                 // choose random decrement amount between 1 and 100
    }
    ASSERT_GT(collection->size(), dec);         // ensure amount to decrement is less than size of collection

    collection->resize(collection->size() - dec);
    EXPECT_LT(collection->size(), size);
}
```

# Throw exception

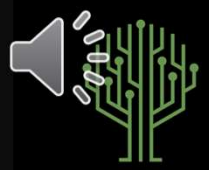
```
// TODO: Create a test to verify the std::out_of_range exception is thrown when calling at()
// with an index out of bounds
// NOTE: This is a negative test
TEST_F(CollectionTest, ThrowsOutOfRangeError){
    add_entries(rand() % 100 + 1);          // ensure collection is not empty
    ASSERT_GT(collection->size(), 0);

    EXPECT_THROW(collection->at(collection->size()), std::out_of_range);
    EXPECT_THROW(collection->at(-1), std::out_of_range);
}
```

# Search for removed value

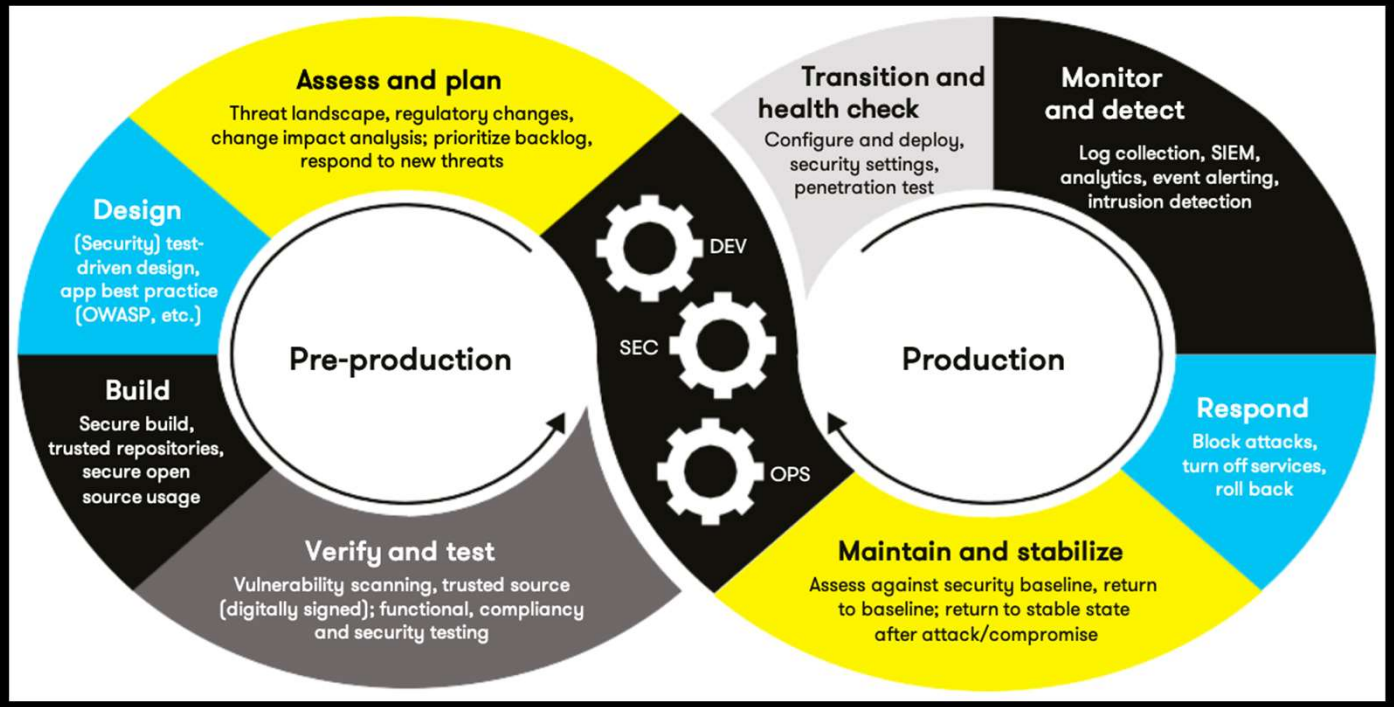
```
//push a unique value, pop collection, search for value (should not find)
TEST_F(CollectionTest, FindAfterPop) {

    collection->push_back(0);               // ensure starting values are unique
    collection->push_back(1);
    collection->push_back(5);
    collection->push_back(10);

    ASSERT_EQ(collection->size(), 4);

    collection->pop_back();

    ASSERT_EQ(collection->size(), 3);

    auto it = std::find(collection->begin(), collection->end(), 10);
                                            // should not not find, so it = collection->end()

    EXPECT_EQ(it, collection->end());

}
```

# Results

```
Running main() from D:\a\_work\1\s\googletest\googletest\src\gtest_main.cc
[==========] Running 4 tests from 1 test case.
[----------] Global test environment set-up.
[----------] 4 tests from CollectionTest
[ RUN      ] CollectionTest.CanAddFiveValuesToVector
[       OK ] CollectionTest.CanAddFiveValuesToVector (0 ms)
[ RUN      ] CollectionTest.ResizeDecrease
[       OK ] CollectionTest.ResizeDecrease (0 ms)
[ RUN      ] CollectionTest.ThrowsOutOfRangeError
[       OK ] CollectionTest.ThrowsOutOfRangeError (0 ms)
[ RUN      ] CollectionTest.FindAfterPop
[       OK ] CollectionTest.FindAfterPop (0 ms)
[----------] 4 tests from CollectionTest (1 ms total)

[----------] Global test environment tear-down
[==========] 4 tests from 1 test case ran. (2 ms total)
[  PASSED  ] 4 tests.
```



# AUTOMATION SUMMARY




Green Pace

# TOOLS

- CODESonar
- Sonar Cube
- PVS-Studio
- SpotBugs
- Polyspace BugFinder

# RISKS AND BENEFITS

| Act Now | Respond Later |
| --- | --- |
| Threat is out there | Nothing has happened yet |
| Cost over run | Cost more to fix after the fact |
| Delayed product delivery | Takes more time to respond to incident |

# RECOMMENDATIONS

GAPS

- Policy and standards cannot address every bit of code
- Who is responsible for security upon deployment?
- Education and awareness
- Need data recovery plan and incident response guide

# CONCLUSIONS

- Splunk has identified 50 of the top cybersecurity threats.  Many more exist.  Continual vigilance and awareness of new trends is a must.
- Consider using a third party to provide security as a service for consistency across products and clients.
- Educate employees and clients, especially as new threats and procedures emerge
- Create a disaster recovery plan

Green Pace

# REFERENCES


CODESonar. (n.d.). *CODESonar Datasheet.* Retrieved from codesonar.com: https://codesecure.com/wp-content/uploads/2024/12/CodeSonar-8.3-Datasheet.pdf

Crowdstrike. (2024). *Global Threat Report.* Retrieved from go.crowdstike.com: https://go.crowdstrike.com/rs/281-OBQ-266/images/GlobalThreatReport2024.pdf

Cybersecurity Guide Contributors. (2024, 04 15). *Safeguarding the environment: Cybersecurity in environmental protection*. Retrieved from cybersecurityguide.org: https://cybersecurityguide.org/industries/environmental-protection/

European Union Agency for Cyber Security. (2024, 06). *ENISA Threat Landscape.* Retrieved from ensia.europa.eu: https://www.enisa.europa.eu/sites/default/files/2024-11/ENISA%20Threat%20Landscape%202024_0.pdf

Free Software Foundation, Inc. (2024). *GCC, the GNU Compiler Collection.* Retrieved from gcc.gnu.org: https://gcc.gnu.org

Kreisa, M. (2024, 07 24). *How to create a disaster recovery plan*. Retrieved from pdq.com: https://www.pdq.com/blog/how-to-create-a-disaster-recovery-plan/

MathWorks. (2024). *Polyspace Bug Finder.* Retrieved from mathworks.com: https://www.mathworks.com/help/bugfinder/index.html

PVS-Studio. (2024, 11 15). *PVS-Studio manual.* Retrieved from pvs-studio.com: https://pvs-studio.com/en/docs/

SonarQube. (2024). *SonarQube for IDE Documentation.* Retrieved from sonarsource.som: https://docs.sonarsource.com/sonarqube-for-ide/intellij/

Splunk. (2023). Top 50 Cybersecurity Threats. Splunk. Retrieved from https://www.splunk.com/en_us/pdfs/gated/ebooks/top-50-cybersecurity-threats.pdf

SpotBugs. (2024). *SpotBugs.* Retrieved from spotbugs.github.io: https://spotbugs.github.io/


Green Pace