EUSEC
EU SECURITY CERTIFICATION

EU-SEC The European Security Certification Framework

ARCHITECTURE AND TOOLS

FOR AUDITING

Fraunhofer
AISEC

Objective & Motivation

- Objective
  - develop and implement a unified way to configure and deploy existing tools to support standardized evaluation of controls
- Motivation
  - Semantics of measurement results produced by measurement techniques have to be comparable, regardless of a measurement technique's
  - Exemplary objective (SQO): Cloud Service's endpoints only use strong TLS cipher suites.
  - Underlying general question: What do we measure?
  - Core challenge: Unambiguous definition of evidence production and measurement to provide for comparability of evidence and measurement results generated by continuous security audit tools

Approach

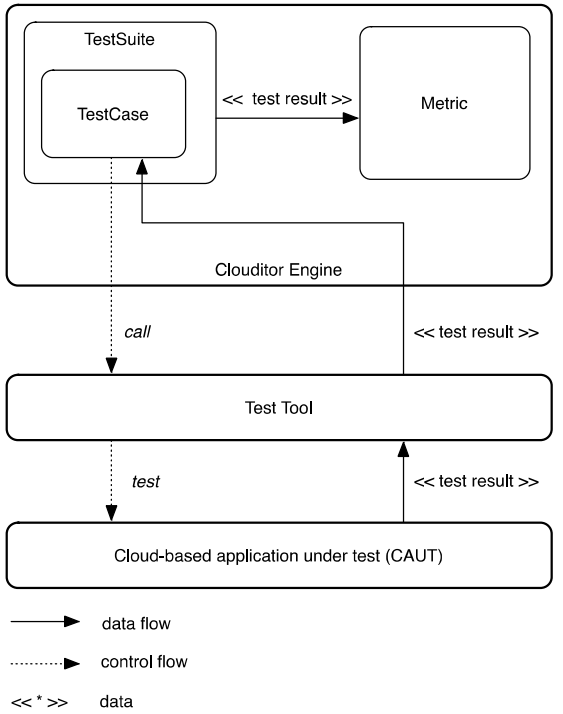
- Development of a domain-specific language (DSL)
  - allows for unified, rigorous representation of configurations of continuous security audits tools
- Scope
  - Test-based security audits (i.e., test-based evidence and measurement production)
- DSL Engineering Process (according to Mernik et al.):
  - Decision: Substantiate the choice to build a DSL.
  - Analysis: Identify common domain constructs which are needed to configure continuous security audit tools.
  - Design: Given the domain constructs, DSL is designed using a suitable formal grammar.
  - Implementation: Implement language and develop tool-specific code generators (e.g. for Clouditor) which compile the target configuration language (i.e., translate from DSL to specific configuration language)

Solution Overview

- Decision: Why build a DSL?
  - General motivation: Provide comparability of evidence and measurement produced by continuous security audit tools
  - guide the development of future continuous security audit tools
  - having tools conform with a common set of domain concepts defined for continuous, test-based evidence and measurement production
  - using formal grammar to define DSL: conforming with the domain concepts is not merely informal but can be enforced through code generators
- Link to T3.1 and T3.3
  - T3.1: unified configuration language allows to define candidate configurations, that is, templates for configurations of continuous security audit tools and map them to corresponding controls
  - T3.3: Instances of evidence and measurement results have to contain the configuration of a continuous security audit tool which was used to produce it

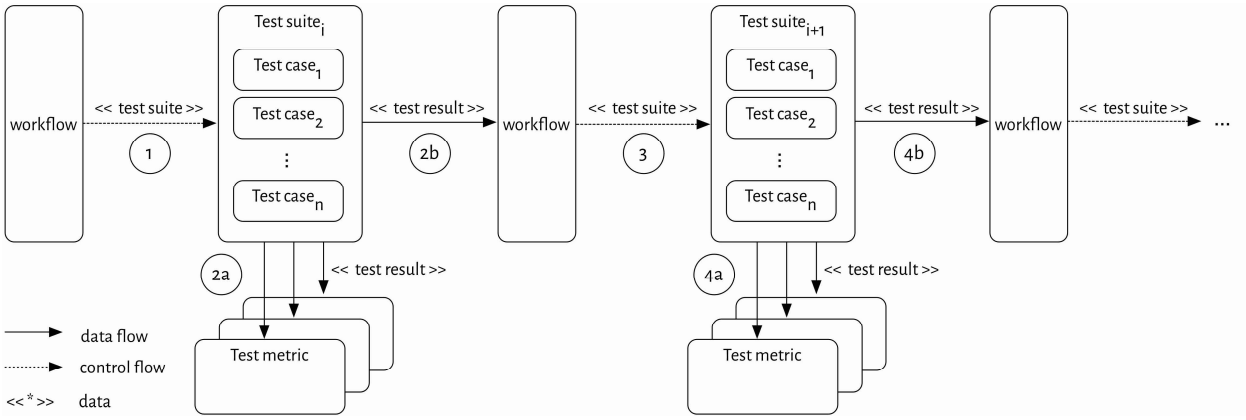
Solution Overview

- Analysis: Building blocks of contiuous test-based security audits
  - Test cases: primitive
    - Example: perform TLS handshake with endpoint and compare supported cipher suites with predefined whitelist
  - Test suites: combine test cases, schedules singular test execution
    - Example: Check supported cipher suites every 30 minutes
  - Workflow: test suite to be run next
    - Example: If cipher suites not contained in the whitelist are supported, check supported cipher suites every 10 seconds



Solution Overview

- Analysis: Building blocks of contiuous test-based security audits
  - Test metrics: produce measurement results based on test (suite) results
    - Example: Computes a the time a the endpoint of the cloud service supported vulnerable cipher suites
  - Preconditions: Check assumptions about environment before or in parallel to test execution
    - Example: Ping before test execution and only execute test if ping succeeded
  - Process



Solution Overview

EUSEC
EU SECURITY CERTIFICATION

- Design: Given the domain constructs, DSL is designed using a suitable formal grammar
  - Building blocks serve as input to design language constructs
  - Extended-Backus-Naur Form (EBNF) is used to define the context-free grammar which generates ConTest
- Implementation: Implement language and develop tool-specific code generators
  - xText: open source framework to support the development and implementation of domain-specific languages
  - provides various features such as parser generation, code generator or interpreter
  - integrates with the Eclipse IDE and providing editor features such as syntax coloring, code completion and source code navigation
  - uses a proprietary language to specify the grammar of a DSL (but very similar to EBNF)

Solution Overview

- Example configuration written in EBNF to continuously check supported cipher suites of a cloud service endpoint

```
TestID d9ecdea3-e9da-4f78-b041-69b7509e3462
TestName PortConTest
TestDescription "Continuously tests whether the interfaces of a cloud service component are securely configured."
Testmetrics{
  TestMetricID c5fddea3 {
    TestMetricName "InsecureConfigurationCounterMetric"
    TestMetricModule "basicResultCounter"
    Description "Counts the occurences of failed tests for secure interface configurations."
  }
  TestMetricID a5fddea3 {
    TestMetricName "YearlyInsecureConfigurationMetric"
    TestMetricModule "cummulativeFailedPassedSequenceDuration"
    Description "Calculates the ratio between measured insecure interface configuration duration since
                 test start and 31557600 seconds in a year of 365.25 days."
  }
  TestMetricID b5fddea3 {
    TestMetricName "DailyInsecureConfigurationMetric"
    TestMetricModule "cummulativeFailedPassedSequenceDuration"
    Description "Calculates the daily insecure interface configuration duration starting from 00:00 am to 23:59."
  }
}

Workflow {
    WorkflowID "PortTestWorkflow"
    WorkflowName "PortTestwithICMPPreconditionWorkflow"
    WorkflowModule "BasicIterator"
    BoundTestsuites {
      TestSuiteID "PortTestNampTestSuite" {
        TestSuiteName "SecureInterfaceConfigurationTestSuite"
          NumberOfMaxIteration infinite
          IntervalBetweenTests  {
              randomizeIntervalDuration [30,180]
              }
          Offset 15
          Timeout 300
          BoundTestCases {
            TestCaseID PingTest {
                TestCaseName "PreConditionPingTestCase"
                  TestCaseModule "Ping"
                  Order 1
                  InputParameters  {
                   <"count": 10>,
                   <"host": "10.244.250.9">
                  }
                  AssertParameters {
                   <"round-trip-avg-$lte":"50 ms">,
                   <"round-trip-sd-$lte":"25 ms">
                  }
            }

            TestCaseID PortTest {
                TestCaseName "PortTestCase"
                  TestCaseModule "nmap"
                  Order 1
                  InputParameters {
                   <"host" :"10.244.250.9">
                  }
                  AssertParameters {
                  <"WhiteListedPorts$eq": [80,443,486,993]>,
                  <"BlackListedPorts$eq": [21,22,25]>
                  }
          }
        }
      }
    }
}
```

Solution Overview

- Example configuration written in ConText to continuously check supported cipher suites of a cloud service endpoint (implemented using xText)

```
TestID d9ecdea3-e9da-4f78-b041-69b7589e3462
TestName "PortConTest"
Description "Continuously tests whether the interfaces of a cloud service component are securely configured"

Testmetrics {
        TestMetricID c5fddea3 {
        TestMetricName "InsecureConfigurationCounterMetric"
        TestMetricModule "basicResultCounter"
        Description "Counts the occurences of failed tests for secure interface configurations."
        }

        TestMetricID a5fddea3 {
        TestMetricName "YearlyInsecureConfigurationMetric"
        TestMetricModule "cummulativeFailedPassedSequenceDuration"
        Description "Calculates the ratio between measured insecure interface configuration duration since test start and 31557600 seconds in a year
                of 365.25 days."
        }

        TestMetricID b5fddea3 {
            TestMetricName "DailyInsecureConfigurationMetric"
            TestMetricModule "cummulativeFailedPassedSequenceDuration"
            Description "Calculates the daily insecure interface configuration duration starting from 00:00 am to 23:59."
        }
}

Testcases {
    TestCaseID PingTest
    TestCaseName "PreConditionPingTestCase"
    TestCaseModule "Ping"
    Order 1
    InputParameters {
        <"count":10>,
        <"host":"10.244.250.9">
    }
    AssertParameters {
        <"round-trip-avg-$lte":"75 ms">,
        <"round-trip-sd-$lte":"25 ms">
    }

    TestCaseID PortTest
    TestCaseName "PortTestCase"
    TestCaseModule "nmap"
    Order 1
    InputParameters {
        <"host":"10.244.250.9">
    }
    AssertParameters {
        <"WhiteListedPorts$eq": [80,443,486,993]>,
        <"BlackListedPorts$eq": [21,22,25]>
    }
}

TestSuites
{
    TestSuiteID PortTestNampTestSuite
    {
        TestSuiteName "SecureInterfaceConfigurationTestSuite"
        BoundTestCases {PingTest, PortTest}
        NumberOfMaxIteration infinite
        IntervalBetweenTests {
            randomizedInterval [30,180]
        }
        Offset 15
        Timeout 300
        }
    }
    Workflow {
        WorkflowID PortTestWorkflow
        WorkflowName "PortTestwithICMPPreconditionWorkflow"
        WorkflowModule "BasicIterator"
        BoundTestSuites {PortTestNampTestSuite}
}
```

Solution Overview

- Generated example configuration to continuously check supported cipher suites of a cloud service endpoint
  - Tool specific YAML configuration file
  - Configuration used for Clouditor

```yaml
name: PortConTest
id: d9ecdea3-e9da-4f78-b041-69b7509e3462
description: Continuously tests whether the interfaces of a cloud service component are securely configured

metrics:
  - class: basicResultCounter
    name: InsecureConfigurationCounterMetric
    description: Counts the occurences of failed tests for secure interface configurations.
  - class: cummulativeFailedPassedSequenceDuration
    name: YearlyInsecureConfigurationMetric
    description: Calculates the ratio between measured insecure interface configuration duration since test start and 31557600 seconds in a year
        of 365.25 days.
  - class: cummulativeFailedPassedSequenceDuration
    name: DailyInsecureConfigurationMetric
    description: Calculates the daily insecure interface configuration duration starting from 00:00 am to 23:59.

testCases:
  PreConditionPingTestCase:
    '@id': PingTest
    order: 1
    count: 10
    host: 10.244.250.9
    round-trip-avg-$lte: 75 ms
    round-trip-sd-$lte: 25 ms

  PortTestCase:
    '@id': PortTest
    order: 1
    host: 10.244.250.9
    WhiteListedPorts$eq: [80, 443, 486, 993]
    BlackListedPorts$eq: [21, 22, 25]

workflow:
  class: BasicIterator
  name: PortTestwithICMPPreconditionWorkflow
  testSuites:
    PortTestNampTestSuite:
    name: PortTestNampTestSuite
    label: SecureInterfaceConfigurationTestSuite
    randomized: true
    iteration: -1
    interval: [30,180]
    offset: 15
    timeout: 300
    testCases: ['@ref': PingTest,'@ref': PortTest]
```