EUROPEAN SECURITY CERTIFICATION FRAMEWORK

D3.5

INTEGRATION FRAMEWORK
– FINAL DOCUMENTATION

VERSION 1.0

PROJECT NUMBER: 731845
PROJECT TITLE: EU-SEC

DUE DATE: 31/12/2018
DELIVERY DATE: 24/12/2018

AUTHOR:
Philipp Stephanow-Gierach
Christian Banse

PARTNERS CONTRIBUTED:
Fabasoft, MFSR, CSA, SixSq

DISSEMINATION LEVEL:*
PU

NATURE OF THE DELIVERABLE:**
R

INTERNAL REVIEWERS: SI-MPA, NIXU

*PU = Public, CO = Confidential
**R = Report, P = Prototype, D = Demonstrator, O = Other

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under grant agreement No 731845
EXECUTIVE SUMMARY

This deliverable describes an integration framework for the tools needed to implement continuous security audits supporting cloud service certification. This framework is based on three pillars: The first one consists of describing the interaction of existing techniques which are available as background in the EU-SEC project. The result is a tool chain where each component is based on the specifications described in Deliverable 3.1, 3.2 and 3.3.

The second pillar of the integration framework consists of a risk-driven process describing how to integrate the tool chain with existing cloud services. The steps of this risk-driven integration process include selecting a global integration strategy, discovering cloud service, deriving feasible measurement techniques, selecting of suitable metrics, deploying components of the tool chain, and adapting measurement techniques to changes of the cloud service under audit at runtime. Example application of the integration process to produce evidence and measurement results on application level as well as on platform level are described.

The third pillar of the integration framework aims at quantifying inaccuracy in measurement results produced by continuous test-based measurement techniques because erroneous results undermine the trust placed in objective evaluation and resulting claims. To that end, a method is presented which permits to evaluate accuracy and precision of measurement results which allows comparing alternative techniques as well as alternative technique's configurations. An example application of this approach is demonstrated where a cloud provider is given a set of candidate configurations for a particular test-based technique and selects the most suited one.
DISCLAIMER

The information and views set out in this publication are those of the author(s) and do not necessarily reflect the official opinion of the European Communities. Neither the European Union institutions and bodies nor any person acting on their behalf may be held responsible for the use which may be made of the information contained therein.

© Copyright in this document remains vested with the EU-SEC Consortium.
<table>
<thead>
<tr>
<th>ACRONYMS</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>AWS</td>
<td>Amazon Web Services</td>
</tr>
<tr>
<td>CAIQ</td>
<td>Consensus Assessments Initiative Questionnaire</td>
</tr>
<tr>
<td>CCM</td>
<td>Cloud Control Matrix</td>
</tr>
<tr>
<td>CMIS</td>
<td>Content Management Interoperability Services</td>
</tr>
<tr>
<td>CRIME</td>
<td>Compression Ratio Info-leak Made Easy</td>
</tr>
<tr>
<td>CSA</td>
<td>Cloud Security Alliance</td>
</tr>
<tr>
<td>CSP</td>
<td>Cloud Service Provider</td>
</tr>
<tr>
<td>CTP</td>
<td>Cloud Trust Protocol</td>
</tr>
<tr>
<td>DDoS</td>
<td>Distributed Denial of Service</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain-specific language</td>
</tr>
<tr>
<td>EBS</td>
<td>Amazon Elastic Block Storage</td>
</tr>
<tr>
<td>EC2</td>
<td>Amazon Elastic Compute Cloud</td>
</tr>
<tr>
<td>EU-SEC</td>
<td>European Security Certification Framework</td>
</tr>
<tr>
<td>IaaS</td>
<td>Infrastructure-as-a-Service</td>
</tr>
<tr>
<td>IEC</td>
<td>International Electrotechnical Commission</td>
</tr>
<tr>
<td>ISO</td>
<td>International Organization for Standardization</td>
</tr>
<tr>
<td>KMS</td>
<td>AWS Key Management Service</td>
</tr>
<tr>
<td>RDS</td>
<td>Amazon Rational Database Service</td>
</tr>
<tr>
<td>S3</td>
<td>Amazon Simple Storage Service</td>
</tr>
<tr>
<td>SLO</td>
<td>Service Level Objective</td>
</tr>
<tr>
<td>SOAP</td>
<td>Simple Object Access Protocol</td>
</tr>
<tr>
<td>SQO</td>
<td>Service Qualitative Objective</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>SSL</td>
<td>Secure Sockets Layer</td>
</tr>
<tr>
<td>TLS</td>
<td>Transport Layer Security</td>
</tr>
<tr>
<td>URL</td>
<td>Uniform Resource Locator</td>
</tr>
<tr>
<td>VDE</td>
<td>Virtual Development Environment</td>
</tr>
<tr>
<td>VM</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>WebDAV</td>
<td>Web-based Distributed Authoring and Versioning</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS

1  INTRODUCTION .......................................................................................................................... 11
1.1  SCOPE AND OBJECTIVE ................................................................................................. 13
1.2  WORKING PACKAGE DEPENDENCIES ............................................................................ 14
1.3  ORGANISATION OF THE DELIVERABLE ........................................................................ 15
2  TOOL CHAIN ......................................................................................................................... 16
2.1  CLOUDITOR ..................................................................................................................... 16
2.2  STARWATCH ....................................................................................................................... 17
2.2.1  INTEGRATION CERTIFICATION TARGETS ............................................................. 18
2.2.2  UPDATING CONTINUOUS ASSESSMENTS .............................................................. 23
2.2.3  USER INTERFACE ....................................................................................................... 24
2.3  SLIPSTREAM ....................................................................................................................... 25
2.3.1  USERS AND BENEFITS ............................................................................................. 26
2.4  TOOL CHAIN: INTERACTION BETWEEN COMPONENTS ............................................. 28
3  INTEGRATING CONTINUOUS SECURITY AUDITS ............................................................ 30
3.1  OVERVIEW ......................................................................................................................... 30
3.2  INTEGRATION PROCESS .................................................................................................... 31
3.2.1  STEP 1: SELECT GLOBAL INTEGRATION STRATEGY ........................................... 31
3.2.2  STEP 2: DEPLOYMENT OF TOOL CHAIN .................................................................. 37
3.2.3  STEP 3: DISCOVER CLOUD SERVICE ...................................................................... 38
3.2.4  STEP 4: DERIVE FEASIBLE MEASUREMENT TECHNIQUES .................................. 39
3.2.5  STEP 5: SELECT SUITABLE METRICS ..................................................................... 41
3.2.6  STEP 6: START EXECUTION OF MEASUREMENTS .................................................... 44
3.2.7  STEP 7: ADAPT MEASUREMENT TECHNIQUES .................................................... 44
4  TECHNICAL INTEGRATION WITH CLOUD SERVICES ..................................................... 46
4.1  APPLICATION LEVEL INTEGRATION ............................................................................... 46
4.1.1  ENVIRONMENTS ....................................................................................................... 46
4.1.2  ACCESSING AUDIT DATA ........................................................................................... 47
4.2  PLATFORM LEVEL INTEGRATION ....................................................................................... 50
4.2.1  ENVIRONMENT ........................................................................................................... 50
4.2.2  PLATFORM APIs ......................................................................................................... 51
4.2.3  EXAMPLE TEST-BASED MEASUREMENTS .............................................................. 51
5  EVALUATION OF CONTINUOUS TEST-BASED MEASUREMENT TECHNIQUES ............... 53
5.1  BACKGROUND .................................................................................................................... 53
### Table of Contents

- 5.1.1 UNIVERSAL METRICS FOR TEST-BASED MEASUREMENT TECHNIQUES ........................................54
- 5.1.2 ACCURACY AND PRECISION ..........................................................58
- 5.2 OVERVIEW OF THE EVALUATION PROCESS ........................................64
- 5.3 SECURITY CONTROL VIOLATION ..........................................................65
- 5.3.1 CONTROL VIOLATION SEQUENCE ..................................................65
- 5.3.2 CONTROL VIOLATION DESIGN .........................................................66
- 5.3.3 STANDARDIZING CONTROL VIOLATION EVENTS ..............................67
- 5.4 ACCURACY AND PRECISION MEASURES ..............................................68
- 5.4.1 BASIC-RESULT-COUNTER ...............................................................68
- 5.4.2 FAILED-PASSED-SEQUENCE-COUNTER ........................................77
- 5.4.3 FAILED-PASSED-SEQUENCE-DURATION ...........................................83
- 5.4.4 CUMULATIVE-FAILED-PASSED-SEQUENCE-DURATION ......................89
- 5.5 IMPLEMENTATION AND EXAMPLE EVALUATION ....................................92
- 5.5.1 SETUP AND ENVIRONMENT .............................................................92
- 5.5.2 CONTINUOUSLY TESTING SECURE COMMUNICATION CONFIGURATION ..........93
- 6 CONCLUSION .....................................................................................106
- 7 REFERENCES ....................................................................................108
LIST OF TABLES

TABLE 6-1 SUMMARY OF CONTROL VIOLATION SEQUENCE STATISTICS FOR TLSTEST ..........95
TABLE 6-2 SUMMARY OF TEST STATISTICS OF TLSTEST .........................................................99
TABLE 6-3: EVALUATION OF TLSTEST TO TEST SECURE COMMUNICATION CONFIGURATION OF SAAS\textsuperscript{OS} BASED ON THE BASIC RESULT COUNTER (BRC) TEST METRIC ...................... 101
TABLE 6-4: EVALUATION OF TLSTEST TO TEST SECURE COMMUNICATION CONFIGURATION OF SAAS\textsuperscript{OS} BASED ON THE FAILED-PASSED-SEQUENCE COUNTER (FPSC) TEST METRIC ................................................................................................................................. 102
TABLE 6-5: EVALUATION OF TLSTEST TO TEST SECURE COMMUNICATION CONFIGURATION OF SAAS\textsuperscript{OS} BASED ON THE FAILED-PASSED-SEQUENCE DURATION (FPSD) TEST METRIC ................................................................................................................................. 102
TABLE 6-6: EVALUATION OF TLSTEST TO TEST SECURE COMMUNICATION CONFIGURATION OF SAAS\textsuperscript{OS} BASED ON THE CUMULATIVE FAILED-PASSED-SEQUENCE DURATION (CFPSD) TEST METRIC ................................................................................................................................. 104
LIST OF FIGURES

FIGURE 1-1: DEPENDENCIES OF TASK 3.4 ........................................................................................................15
FIGURE 2-1 TOOLS OF THE CLOUDITOR ECOSYSTEM ................................................................................16
FIGURE 2-2 OVERVIEW OF CLOUDITOR’S ENGINE MAIN COMPONENTS (WITH EXTERNAL TEST TOOL) .................................................................................................................................17
FIGURE 2-3 HIGH LEVEL OVERVIEW OF THE MULTI-CLOUD APPLICATION MANAGEMENT OFFERED BY SLIPSTREAM ......................................................................................................................25
FIGURE 2-4 FULL APPLICATION LIFECYCLE MANAGEMENT THROUGH SLIPSTREAM .................................26
FIGURE 2-5. HIGH-LEVEL VIEW ON TOOL INTERACTION ....................................................................................28
FIGURE 3-1: INTEGRATION PROCESS OF TOOL CHAIN TO SUPPORT CONTINUOUS SECURITY AUDITS OF CLOUD SERVICES ...........................................................................................................31
FIGURE 3-2: EXTRACT OF SERVICE DESCRIPTION FOR IAAS PROVIDED BY OPENSTACK ........38
FIGURE 5-1 EXEMPLARY FAILED-PASSED-SEQUENCE (fps) BASED ON BASIC TEST RESULTS (br) 
............................................................................................................................................................................55
FIGURE 5-2 EXAMPLE DEFINITION FOR UNIVERSAL TEST METRIC fpsD ..........................................................56
FIGURE 5-3 AVAILABLE OPTIONS TO DEFINE FAIL-PASS-SEQUENCE-DURATION (fpsD) IF |fps| > 2 .......................................................................................................................................................57
FIGURE 5-4 EXPERIMENTAL EVALUATION OF THE ACCURACY AND PRECISION OF TEST-BASED MEASUREMENT TECHNIQUES ..............................................................................................................65
FIGURE 5-5 SEQUENCE OF CONTROL VIOLATION EVENTS cve ........................................................................66
FIGURE 5-6 TRUE NEGATIVE BASIC TEST RESULT (brTN) ..............................................................................69
FIGURE 5-7 TRUE POSITIVE BASIC TEST RESULT (brTP) ....................................................................................70
FIGURE 5-8 FALSE NEGATIVE BASIC TEST RESULT (brFN) ...............................................................................71
FIGURE 5-9: FALSE POSITIVE BASIC TEST RESULT (brFP) ...............................................................................71
FIGURE 5-10 FALSE POSITIVE BASIC TEST RESULT (brCFP) .............................................................................72
FIGURE 5-11 PSEUDO TRUE NEGATIVE BASIC TEST RESULT (brCPTN) ..........................................................73
FIGURE 5-12 PSEUDO TRUE POSITIVE BASIC TEST RESULT (brPTN) ............................................................73
FIGURE 5-13 TRUE NEGATIVE FAILED-PASSED-SEQUENCE (fpsTN) ...............................................................78
FIGURE 5-14 FALSE NEGATIVE fps ....................................................................................................................79
FIGURE 5-15 FALSE NEGATIVE FAILED-PASSED-SEQUENCE (fpsFN) WITH TRUE NEGATIVE AND FALSE NEGATIVE BASIC TEST RESULT (brTN & brFN) ...............................................................79
FIGURE 5-16 FALSE NEGATIVE FAILED-PASSED-SEQUENCE (fpsFN) WITH FALSE POSITIVE BASIC TEST RESULT (brFP) ........................................................................................................................80
FIGURE 5-17 FALSE POSITIVE fps ....................................................................................................................80
FIGURE 5-18 TRUE NEGATIVE FAILED-PASSED-SEQUENCE-DURATION (fpsTN) WHICH OVERESTIMATES TOTAL DURATION OF $c_{vei}$ AND $c_{vei} + 1$..........................................................84

FIGURE 5-19 TRUE NEGATIVE FAILED-PASSED-SEQUENCE-DURATION (fpsTN) WHICH UNDERESTIMATES DURATION OF $c_{vei}$..........................................................84

FIGURE 5-20 TRUE NEGATIVE FAILED-PASSED-SEQUENCE-DURATION (fpsTN) WITH $efpsD_{preTN} > 0$ AND $efpsD_{postTN} > 0$..........................................................85

FIGURE 5-21 FALSE NEGATIVE FAILED-PASSED-SEQUENCE-DURATION (fpsDFN).............85

FIGURE 5-22 CAPTION FALSE POSITIVE FAILED-PASSED-SEQUENCE-DURATION (fpsDFP)...86

FIGURE 5-23 RELATIVE DURATION ERROR OF FPSD ($efpsD_{relTN}$) OF TLTEST$^{[0,10]}$, TLTEST$^{[0,30]}$, AND TLTEST$^{[0,60]}$ ...............................................................................................................................................98
1 INTRODUCTION

This deliverable serves as an update to the previous version of the integration framework, which was already submitted as Deliverable 3.4 – Integration Framework V1.0 and replaces said deliverable. The following table highlights the high-level changes between Deliverable 3.4 and 3.5:

- Merged contents of previous Section 2.2 “CTP Server” into the sections of Clouditor and STARWatch. Clouditor is now responsible for evaluating control objectives.
- Section 2.2 “STARWatch” now describes the common data format used for exchanging certificate targets as well as the interface used by Clouditor to STARWatch.
- In Section 2.4 “Tool chain: interaction between components”:
  - Added reference to the data format, defined in Deliverable 3.1
  - Clarified the use of Clouditor for the objective evaluation
  - Clarified the use of STARWatch for claim and certificate management
  - Clarified the use of Slipstream as evidence store using CIMI
- Combined the previous Chapters 4 and 5 into new Chapter 4 “Technical Integration with Cloud Services”, with two sections 4.1 and 4.2 describing Application Level and Platform Level Integration, respectively.
  - The aspects of applying the integration process (previously 4.2 and 5.2) are now addressed by Deliverable 5.2 in WP5.
  - The Application Level Integration section was reworked to be more focused on the Continuous Audit API, using the Fabasoft Cloud implementation as an example, in order to address a broader spectrum of applications.

Integration of continuous security audits with existing cloud services to support continuous certification requires to consider the following key aspects:
INTEGRATION OF TOOLS NEEDED TO ENABLE CONTINUOUS SECURITY AUDITS

Multiple applications are required to implement continuous security audits of cloud services. As already pointed out in Deliverable 3.1, 3.2 and 3.3, these applications include: Objective evaluation application (Deliverable 3.1), continuous measurement techniques (Deliverable 3.2) as well as evidence stores (Deliverable 3.3). These applications have to interact in a well-defined manner to enable continuous security audits, that is, they have to be integrated with each other to implement the tool chain required for continuous cloud security audits.

INTEGRATION OF THE TOOL CHAIN WITH EXISTING CLOUD SERVICES

In order for the tool chain to become meaningful, it has to interact with existing cloud services in a well-defined way such that evidence and measurement results are produced (and stored) supporting the validation of controls. Integrating the tool chain with a cloud service is not confined to integrating the continuous measurement technique used to produce evidence and compute measurement results. It also has to address questions such as: Where to host the evidence store? How to handle changes of the configuration or composition of cloud service under audit? Where to host the claim store?

EVALUATION OF ACCURACY AND PRECISION OF MEASUREMENT RESULTS

The tool chain continuously, i.e., automatically and repeatedly produces and stores measurement results to support validation of controls of security certificates. Inaccurate results undermine both cloud provider’s and customer’s trust: On the one hand, measurement results that incorrectly indicate satisfaction of a control erode customer’s trust. On the other hand, cloud service providers may dispute results incorrectly suggesting that controls are not fulfilled. Therefore, it is essential to evaluate the accuracy and precision of measurement results produced by continuous test-based measurement techniques, that is, how close are produced results to their true values?

Consider, as an example, the following extract of control TVM-02: Threat and Vulnerability Management Vulnerability / Patch Management of CSA’s CCM (1):

“Policies and procedures shall be established, and supporting processes and technical measures implemented, for timely detection of vulnerabilities within organizationally-owned or managed applications, infrastructure network and system components (e.g., network vulnerability assessment, penetration testing) [...].”.
One possibility to produce measurement results supporting validation of this control consists of a test-based technique which executes a vulnerability scanner every ten minutes and checks whether no vulnerability is found. The question is now whether this technique makes mistakes by, e.g., incorrectly suggests that the cloud service under test has no vulnerabilities while it actually has. In this case, it unclear to what extent the produced results can be used to check the control. Does, e.g., the vulnerability scanner only occasionally miss detecting a particular vulnerability or does it never detect it?

1.1 SCOPE AND OBJECTIVE

This deliverable’s main objective is to describe an integration framework for the tool chain which is needed to implement continuous security audits supporting cloud service certification.

As already outlined above, integrating this tool chain first of all requires to integrate existing techniques following the specifications described in Deliverable 3.1, 3.2 and 3.3 with each other. To that end, a subgoal of this deliverable consists of delineating the different components of the tool chain as well as describing their interaction.

Furthermore, the tool chain to implement continuous cloud security audits has to be integrated with existing cloud services. Therefore, another subgoal of this deliverable is to describe a risk-driven integration process which considers different levels of integration, derivation of feasible measurement techniques, selection of and suitable metrics, deployment strategies of the tool chain as well as adaption of measurement techniques at runtime.

Finally, measurement results produced by measurement techniques contain the essential information to determine of a cloud service satisfies a set of SLOs or SQOs. Inaccurate measurement results therefore undermine the trust placed in objective evaluation and resulting claims. Thus, the last subgoal of this deliverable is to provide a method to evaluate the accuracy and precision of measurement results produced by continuous test-based measurement techniques. This method permits to compare alternative techniques as well as alternative technique’s configurations.
1.2 WORKING PACKAGE DEPENDENCIES

The integration framework introduced in this document has dependencies with Task 3.1, 3.2, 3.3 as well as with Task 5.1 of Working Package 5 (see Figure 1-1). Consider Task 3.1 which specifies data structures and protocols used to store and evaluate instances of control objectives. This specification forms the basis for one component of the tool chain described in Section 2 of this deliverable. Furthermore, the data structures defined for objective evaluation in Deliverable 3.1 can serve as the starting point to conduct a risk analysis whose results are required to decide where to host the objective evaluation application during Step 5 Deployment of the tool chain of the risk-driven integration process (see Section 3.2). Similarly, Deliverable 3.3 serves as input to the risk-driven integration process: It define a common data structure to represent evidence, i.e., instances of evidence produced by (test-based) measurement techniques. This data structure depicts the starting point to investigate what additional risk exposure is incurred through storing evidence and this guides the decision where to persist evidence, i.e., where to deploy the evidence store.

Moreover, consider Task 3.2 which develops a domain specific language (DSL) called ConTest which allows rigorously defining continuous test-based measurements. This unified configuration language is crucial when comparing accuracy and precision of alternative test-based techniques as well as alternative configurations because ConTest standardizes configuration representation. This means that ConTest provides a standardized way how to refer to a specific (configuration of a) continuous test-based measurement technique which is necessary for explicit, unambiguous comparison of alternative techniques and alternative techniques' configuration.

Furthermore, the risk-driven integration process of the integration framework (see Section 3.2) presented in this deliverable serves as input to Task 5.1 of Working Package 5 which centers around the preparation of the pilot implementing continuous security audits. Once the pilot has been prepared considering the risk-driven integration process, the process description will be revised according to necessary alterations observed during the pilot.
The remainder of this document is organized as follows: The next section outlines how existing tooling within the EU-SEC project interacts in order to implement continuous cloud security audits supporting cloud certification. Thereafter, Section 3 describes the steps of integrating the tool chain with existing cloud services. Following this integration process, Section 4 describes example integrations to produce evidence on the application level and platform level. These examples are driven by the pilot requirements elicited as part of Task 5.1 of Working Package 5. Section 5 then presents an approach to experimentally evaluate the accuracy and precision of continuous test-based measurement techniques. Finally, Section 6 concludes this deliverable.

1.3 ORGANISATION OF THE DELIVERABLE

Figure 1-1: Dependencies of Task 3.4
2 TOOL CHAIN

This chapter outlines existent tooling and solutions within the EU-SEC project and describes how they interact with each other in order to allow for continuous, i.e., automated and repeated security audits. Sections 2.1 - 2.3 outline tools involved in the tool chain while Section 2.4 describes how these tools interact, thereby composing the tool chain required for continuous cloud security audits.

2.1 CLOUDITOR

The Clouditor toolbox consists of five main components which are shown in Figure 2-1. It can be used to design and execute continuous test-based assurance techniques. The test results serve as input to compute test metrics which, in turn, can be used as evidence to support validation of controls.

The Engine and the Explorer are responsible for continuously executing and adapting assurance techniques. The Simulator and the Evaluator are used prior to deployment, they serve to select techniques and respective configurations which are most suitable to check if a cloud service complies with a particular requirement set. Lastly, the components can be viewed and configured from a Dashboard. Each component is designed as a micro-service and can be deployed in an individual container.

![Figure 2-1 Tools of the Clouditor Ecosystem](image)
In the following, we will only outline components of continuous validation, i.e., the Clouditor Engine and the Clouditor Explorer. The Clouditor Engine implements and deploys test-based assurance techniques. It consists of test suites which comprise test cases, workflows which model dependencies between test suite executions, and metrics which are used to reason about the sequence of results of test suite executions. Figure 2-2 shows a high-level architecture of the Clouditor Engine’s components, including data and control flow.

Discovering a cloud-based application’s interfaces and configuring the selected assurance technique is the task of the Clouditor Explorer. To that end, the Explorer discovers cloud services’ composition and interfaces at runtime as well as automatically generates and adapts test configurations.

Figure 2-2 Overview of Clouditor’s Engine main components (with external test tool)

2.2 STARWATCH

STARWatch is a SaaS application to help organizations manage their compliance with CSA STAR (Security, Trust and Assurance Registry) requirements. STARWatch delivers the content of the Cloud Controls Matrix (CCM) and Consensus Assessments Initiative Questionnaire (CAIQ)

---

1 For further details on the remaining components of the Clouditor Toolbox see https://www.aisec.fraunhofer.de/content/dam/aisec/Dokumente/Publikationen/Studien_TechReports/englisch/Whitepaper_Clouditor_Feb2017.pdf.
in an online editable format, enabling users to manage compliance of cloud services with CSA best practices.

2.2.1 INTEGRATION CERTIFICATION TARGETS

STARWatch will expose an UI interface that will allow a STARWatch administrator\(^2\) to upload a JSON file expressing a certification target, as described in section 2.1. This certification target will be used to create a continuous assessment.

In EU-SEC deliverable 3.1, we defined a JSON data format that is designed to express a certification target, which is essentially:

- A list of high-level requirements (e.g. control objectives) broken down into objectives (SLOs/SQOs or again control objectives).

- An assessment frequency for each objective.

The data format makes a distinction between objectives that are can be assessed automatically (automated_assessment) versus those that require human intervention (assisted_assessment).

We recall below the general structure of this JSON data structure. As a change from deliverable 3.1 we renamed the very first field of the data structure from “certification_objective_id” to “certification_target_id”. The JSON notations used here are the same as in D3.1.

---

\(^2\) More accurately, a user with the rights to create an assessment in his role.
{  "certification_target_id": <string>,  "start_date": <datetime>,  "end_date": <datetime>,  "subject": {    "organisation": <string>,    "service": <string>,    "scope": <string>,  },  "assessment": {    "type": <string>,    "auditor": <string>,    "authority": <string>  },  "requirements": [    {      "requirement_id": <string>,      "requirement_framework": <uri>,      "objectives": [        <assisted_assessment> | <automated_assessment>,        ...      ]    },    ...,    ...,    ]  }
When the certification target JSON file is uploaded, STARWatch will use the data as follows:

<table>
<thead>
<tr>
<th>JSON property</th>
<th>How STARWatch will process the property</th>
</tr>
</thead>
<tbody>
<tr>
<td>certification_target_id</td>
<td>Ignored as input.</td>
</tr>
<tr>
<td>start_date</td>
<td>Start date of the continuous assessment</td>
</tr>
<tr>
<td>end_date</td>
<td>End date of the continuous assessment</td>
</tr>
<tr>
<td>subject.organization</td>
<td>An identifier which must refer to an organization already registered in STARWatch (i.e. Amazon = 17).</td>
</tr>
<tr>
<td>subject.service</td>
<td>The name of the service to be published in the public registry</td>
</tr>
<tr>
<td>subject.scope</td>
<td>A textual description of the service and the scope of the assessment.</td>
</tr>
<tr>
<td>assessment.type</td>
<td>Displayed by STARWatch. Not used in practice.</td>
</tr>
<tr>
<td>assessment.auditor</td>
<td>Displayed by STARWatch. Not used in practice.</td>
</tr>
<tr>
<td>assessment.authority</td>
<td>Displayed by STARWatch. Not used in practice.</td>
</tr>
<tr>
<td>requirements.requirement_id</td>
<td>Will match the corresponding CCM control identifier (e.g. AIS-01).</td>
</tr>
<tr>
<td>requirement.framework</td>
<td>Set to &quot;<a href="https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3-0-1/">https://cloudsecurityalliance.org/download/cloud-controls-matrix-v3-0-1/</a>&quot;</td>
</tr>
<tr>
<td>requirement.objectives</td>
<td>Will only contain automated assessments. (See hereafter)</td>
</tr>
</tbody>
</table>
The structure of an **assisted_assessment** is as follows:

```json
{
    "objective_id": <string>,
    "period": <duration>,
    "type": "assisted",
    "asset_name": <string>,
    "description": <string>
}
```

In the context of the EU-SEC pilot, we will not use “assisted assessments”, so the details of this data structure will be omitted here.

The structure of an **automated_assessment** is as follows. As a modification to the elements specified in EU-SEC deliverable D3.1, a “description” field is added. Moreover, the field named “frequency” was renamed “period” since the original term was semantically incorrect and could create a confusion.

```json
{
    "objective_id": <string>,
    "period": <duration>,
    "type": "automated",
    "description": <string>,
    "asset_name": <string>,
    "metric": <uri>,
    "attribute_name": <string>,
    "measurement_parameters": [
        {
            "name": <string>,
            "type": "number" | "long" | "boolean" | "string",
            "value": <number> | <long> | <string> | <boolean>,
        },
        ...
    ],
    "result_format": [
        {
            "name": <string>,
            "type": "number" | "long" | "boolean" | "string",
        },
        ...
    ],
    "assertion": <string>
}
```
When the certification target JSON file is uploaded, STARWatch will use the data provided in each automated assessment as follows:

<table>
<thead>
<tr>
<th>JSON property</th>
<th>How STARWatch will process the property</th>
</tr>
</thead>
<tbody>
<tr>
<td>objective_id</td>
<td>STARWatch will store this value: tools that submit an update to an assessment will reference this id.</td>
</tr>
<tr>
<td>period</td>
<td>Used by STARWatch to determine the frequency of assessment.</td>
</tr>
<tr>
<td>type</td>
<td>Set to “automated” as defined in D3.1.</td>
</tr>
<tr>
<td>description</td>
<td>This text will displayed in STARWatch as information to the user, explaining in human readable language what the purpose of this objective is.</td>
</tr>
<tr>
<td>asset_name</td>
<td>Used by STARWatch when displaying a continuous assessment.</td>
</tr>
<tr>
<td>metric</td>
<td>Used by STARWatch when displaying a continuous assessment in the registry.</td>
</tr>
<tr>
<td>measurement_parameters</td>
<td>Ignored</td>
</tr>
<tr>
<td>result_format.name</td>
<td>STARWatch will store this value: tools that submit an update to an assessment will reference this name.</td>
</tr>
<tr>
<td>result_format.type</td>
<td>Set to “boolean”: tools submitting an update are expected to check whether the SLO/SQO is achieved, thus reporting “true” or “false”.</td>
</tr>
<tr>
<td>assertion</td>
<td>Displayed by STARWatch. Not used in practice.</td>
</tr>
</tbody>
</table>

The successful upload of a JSON certification target in STARWatch will result in the creation of a **continuous assessment** entry added to the list of assessments that is under the user’s STARWatch license. The corresponding assessment identifier will be provided to the user on the screen (assessment_id). This assessment_id will be necessary for automated tools that will provide updates to the CCM assessment.
The example below shows a continuous assessment with an identifier set to 16.

2.2.2 UPDATING CONTINUOUS ASSESSMENTS

The STARWatch application will expose a method enabling tools to submit updates to STARWatch assessments with the following signature:

PUT /api/v1/continuous_assessment/:assessment_id

The placeholder :assessment_id should be replaced with the actual assessment id that needs to be updated.

This request must be accompanied with an API key provided in the in the "Authorization" request header. The body of the PUT request will have the following structure:

```json
{
    assessment_id: <string>,
    objective_id: <string>,
    result: <boolean>,
    assessed_at: <UTC_time>,
    evidence: [          // optional
        <string>,
        ...
    ]
}
```

<table>
<thead>
<tr>
<th>JSON property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>objective_id</td>
<td>Refers to the objective that is being updated.</td>
</tr>
<tr>
<td><strong>result</strong></td>
<td>A Boolean describing whether or not the objective is fulfilled.</td>
</tr>
<tr>
<td><strong>assessed_at</strong></td>
<td>A timestamp describing when the objective was assessed.</td>
</tr>
<tr>
<td><strong>evidence[]</strong></td>
<td>An array of pointers to supporting evidence. These strings can be URLs or simply identifiers, if the context is sufficiently clear. STARWatch will not perform any checks on these values or publish them in the public registry, but will display them to the service owner.</td>
</tr>
</tbody>
</table>

Upon success, the API endpoint will return an updated version of the automated_assessment described previously, which will contain 4 additional elements for information purposes:

<table>
<thead>
<tr>
<th><strong>JSON property</strong></th>
<th><strong>Description</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>revoked_at</strong></td>
<td>Provides a date after which the objective would cause the certificate to be revoked if not satisfactory update is provided.</td>
</tr>
<tr>
<td><strong>suspended_at</strong></td>
<td>Provides a date after which the objective would cause the certificate to be suspended if not satisfactory update is provided.</td>
</tr>
<tr>
<td><strong>updated_at</strong></td>
<td>A timestamp describing when the objective was updated.</td>
</tr>
<tr>
<td><strong>valid_from</strong></td>
<td>A timestamp describing the beginning of the next update period.</td>
</tr>
</tbody>
</table>

### 2.2.3 USER INTERFACE

Continuous assessments will be displayed in the user interface of STARWatch, as a distinct type of assessment form the existing CAIQ.

At this stage the only elements that the user should be able to edit are the same elements as those that can be submitted in the API defined in section 2.2.1. All other information should be read-only.
2.3 SLIPSTREAM

SlipStream\(^3\) is a multi-cloud application deployment engine and brokerage system that federates any number of clouds and allows users to deploy and manage cloud applications on and across those clouds (Figure 2-3). It is the central management and control behind Nuvla, the SaaS deployment of SlipStream that is managed by SixSq, which is the central access point for users’ cloud resources.

Through Nuvla\(^4\), users can easily automate the deployment and maintenance of their platform, targeting any connected cloud without having to change the application definition.

\(\text{Figure 2-3 High level overview of the multi-cloud application management offered by SlipStream}\)

Leveraging resources from Infrastructure as a Service (IaaS) cloud providers, SlipStream manages cloud applications through the full lifecycle: deployment, configuration, validation, scaling, and termination (Figure 2-4).

---

\(^3\) https://sixsq.com/products-and-services/slipstream/overview

\(^4\) https://sixsq.com/products-and-services/nuvla/overview
SlipStream's essential features include:

- Enterprise App Store built-in: Self-service IT delivered for the enterprise, simplifying application provisioning dramatically;
- Recipe/template/blueprint: Define and execute deployments, based on high-level recipes (script, Puppet, Chef, Ansible, etc.);
- Cloud Broker Enablement: Supports most public and private IaaS;
- Multi-cloud Management: Supports hybrid and multi-cloud deployment scenarios.

2.3.1 USERS AND BENEFITS

Cloud technologies provide real benefits to users and organizations, but they also have their own challenges.

- Incompatible APIs: Make it difficult to move applications from one cloud to another and complicate the simultaneous use of different clouds.
- Opaque VMs: Keeping track of what virtual machines contain (data and services) and managing their updates are difficult.
- Component vs. Application: Most applications comprise multiple layers with numerous individual machines. Cloud services oriented towards single VMs make application management more tedious.

SlipStream addresses these challenges by providing its users with an efficient platform for the management of the full lifecycle of cloud applications.

A number of different types of people within an organization can benefit from SlipStream:
a) Those who are working on different projects and need IT applications and resources – they can benefit from the SlipStream App Store where they can start the applications they need with one click;

b) Those who manage a number of workers taking advantage of cloud resources and want an overview of their resource usage to understand costs and their involving needs – SlipStream provides the ability to monitor resource utilization;

c) Those who develop cloud applications for other people within their organization – they benefit from SlipStream by creating a rich catalog of services that can be automatically and reliably deployed; and

d) Those who manage their own SlipStream installation – they can integrate their own cloud infrastructure into their SlipStream deployment and control what external cloud resources are available to their users.

Read more about possible SlipStream use cases at http://sixsq.com/products/slipstream/usecases/.
2.4 TOOL CHAIN: INTERACTION BETWEEN COMPONENTS

Figure 2-5. High-level view on tool interaction

Figure 2-5 shows a high-level overview of the interaction between tools which are either already existent (see previous sections) or will be developed in the course of EU-SEC. The interaction between the tools works as follows: A continuous test-based measurement technique such as Clouditor uses tests to produce evidence (Step 1). Each test result is stored in the evidence store (Step 3b) where it can later be looked up by a customer or auditor in case of, e.g., disputes. This point will be further detailed in Step 4. Note that only parts of a test result are considered evidence whereas the test result already embodies a decision made on the basis of the information which has been obtained during the test’s execution. The test-based measurement technique applies some function which is referred to as test metric to the test results which it observes, e.g., counts the occurrence of failed tests or the duration of successively failing tests (Step 2, for further details see also Chapter 4 of Deliverable 3.2). The

---

5 In terms of testing terminology, any information which serves as input to well-defined test oracles are considered evidence. This renders each test oracle which forms a part of a test a primitive metric (for further information see Section 4.1.5, Deliverable 3.2).
output of that function is referred to as measurement result. These measurement results are supplied to the objective evaluation application (see Step 3a) which uses rules to reason about the measurement results, e.g., according to the measurement results, has the cloud service been available for at least 99.999% during the last 360 days (Step 4). The data format which provides the specification to implement such an objective evaluation application, can be found in Section 2.2.1. Within the scope of the project, Clouditor will serve as both the tool to produce as well as to evaluate it. The result of applying these rules determines whether a cloud service satisfies a particular control objective derived from some control of a certification scheme. Note that this mapping is based on manually derived expert consensus, i.e., there is no rigorous method available to automatically interpret a control objective. The result of evaluating a control objective is referred to as a claim stating either a controls satisfaction or dissatisfaction at a certain point in time. The claims are forwarded to the claim storage, such as STARWatch where they are persisted (Step 5). In case an authorized party, e.g., a cloud service user, has doubts about the claim or wants to confirm the claim, the customer can inquire the evidence (contained in the atomic the test results) which was used to generate the claim (Step 6). Deployment and management of applications involved in the tool chain, e.g., evidence store and claim store, can be facilitated through SlipStream. Additionally, SlipStream is used as the evidence store itself (Step 3b) using a standardized interface for managing Cloud meta-data, Cloud Infrastructure Management Interface (CIMI), which is described in Deliverable 3.3.
3 INTEGRATING CONTINUOUS SECURITY AUDITS

This chapter describes the steps involved when integrating the tool chain described in the previous chapter with an existing cloud service. The following section provides a high-level overview of the process while Section 3.2 describes each step of the integration process in detail.

3.1 OVERVIEW

Figure 3-1 shows the steps which need to be taken to integrate the tool chain described in Section 2.4 with a cloud service which is sought to be subject to continuous security audits. These steps include:

1. **Select global integration strategy for toolchain:** In the first step, the general integration strategy for the toolchain is selected which is driven by the additional risk which a cloud service provider is willing to tolerate when planning to support continuous security audits.
2. **Deploy tool chain:** Drawing on the general integration strategy, in the second step, the deployment strategy is determined, that is, it is defined where to run certain parts of the continuous security audit tool chain, including: Test-based measurement techniques, objective evaluation, as well as evidence and claim storage. Since the deployment strategy of the tool chain is derived from the global integration strategy, deployment of the tool chain is also risk-driven.
3. **Discover cloud service:** In the third step, the components of the cloud service which is sought to be subjected to continuous security audits are discovered.
4. **Derive feasible measurement techniques:** In the fourth step, feasible evidence production techniques for the discovered cloud service are derived.
5. **Select feasible metrics:** In the fifth step, the measurement results are derived based on the evidence that can be produced for a discovered cloud service.
6. **Start execution of measurements:** In the sixth step, the execution of the measurement techniques is triggered, thereby rendering the tool chain operational.
7. **Adapt measurement techniques at operation time:** In the seventh step, compositional as well as configuration changes of the cloud service under audit are continuously
discovered at operation time of the tool chain. In case of changes, evidence production techniques are adapted accordingly while preserving semantics of computed measurement results.

Figure 3-1: Integration process of tool chain to support continuous security audits of cloud services

3.2 INTEGRATION PROCESS

This section describes the steps of the integration process in detail.

3.2.1 STEP 1: SELECT GLOBAL INTEGRATION STRATEGY

This step determines the global integration strategy of the tool chain.

Note that the discussion of integration variants described hereafter relies on the following assumption: Integrating parts of the tool chain which do not directly interact with the cloud service under audit (i.e., evidence store, claim store and objective evaluation application) as part of the service’s infrastructure provides superior security properties. The rationale behind this is that adding further external environments to run parts of the tool chain leads to a relatively higher increase in attack surface because these other external environments (i.e., infrastructure where tool chain parts can be run) have be communicated with as well as maintained in a secure manner. However, it is important to point out that this assumption does not always have to be true, for example, if the cloud service provider under audit is malicious and attempts to manipulate parts of the tool chain to alter, e.g., measurement results.

RISK-DRIVEN INTEGRATION OF MEASUREMENT TECHNIQUES

Different levels of invasiveness are introduced hereafter which a continuous security audit tool may require to produce evidence as well as measurement results to support the validation of security controls. Recall that a continuous security audit tools can draw on two classes of measurement techniques: Monitoring-based and test-based measurement techniques. The
former use monitoring data as evidence which is produced during productive operation of a
cloud-service. The latter also collects evidence while a cloud-service is productively operating.
Different to monitoring-based methods, however, test-based methods do not passively
monitor operations of a cloud service but actively interact with it through tests.

The level of integration required for evidence and measurement result production is
determined by the changes of the productive environment of the cloud service to be
continuously audited, that is, the required changes of each component involved in productive
service delivery. Hereafter, non-invasive, minimally invasive and invasive integration of
measurement techniques are described.

- **Non-invasive integration**: As the name indicates, this type of integration requires no
  change of the productive environment which is used to operate the cloud service under
  audit. This means that a measurement technique can produce suitable evidence without
  requiring any changes to the cloud service. This type of integration implies that the
  implementation of the measurement technique does not have to be part of the cloud
  service infrastructure but can operate on a remote host, external to the cloud service's
  infrastructure.

  As a basic scenario, consider the endpoint of a SaaS application, i.e., a web site which
  is publicly reachable. In order to automatically produce measurement results as to
  whether this endpoint supports secure communication with its users, no further
  privileges are needed. As a different example, consider a SaaS application to which only
  authorized user have access. In order to automatically assess whether, for example, any
  input fields available to authorized users properly validate user input and thus do not
  possess some SQL injection vulnerability, user level access privileges are required. Still,
  this example measurement technique does not require to change the composition or
  configuration of production environment of the cloud service.

- **Minimally invasive integration**: This type of integration requires to change the
  configuration of the production environment of the cloud service under audit to permit
  the measurement technique to produce measurement results. Similar to non-invasive
  integration, minimally invasive techniques does not have to be deployed and operated
  as part of the cloud service’s infrastructure.

  As an example, consider changing security groups to allow a remote host sending TCP
  segments to a cloud service component, e.g., a virtual machine to check its
  responsiveness. The original security model of the cloud service may not permit some
  components to be accessed from external hosts which are not part of the cloud service's
infrastructure. Therefore, in this example, the configuration of the cloud service under audit has to be altered for the measurement technique to work correctly.

- **Invasive integration**: This type of integration requires to change the composition of or the applications used by a cloud service’s productive environment to allow measurement techniques to produce suitable measurement results. Contrary to non-invasive and minimally invasive integration, invasive integration of measurement techniques implies that at least some parts of techniques’ implementation are integrated with the production environment which is used to operate the cloud service under audit. We can distinguish the following subtypes of invasive integration:

  1. **Compositional changes**: In this case, structural changes to the cloud service composition are needed such as adding a virtual machine or micro service where the measurement technique is deployed and operating on. A classic example of invasive integration through compositional changes are so-called *monitoring agents*, i.e., additional applications deployed on virtual or physical components of the cloud service collecting information such as CPU load.

  2. **Code-level changes**: Here, changes in the form of patches to applications which constitute components of cloud services are needed in order to produce measurement results. Consider, as an example, changing the scheduler of a cloud platform management system such as OpenStack to be able monitor deployments of virtual machines to determine if some machines of a particular user are only using designated hosts, that is, do not to share the underlying hardware with machines of other users.

Changing configuration (i.e., minimally invasive integration) or composition (i.e., invasive integration) of the production environment of the cloud service to be continuously audited may increase the attack surface of the service. Therefore, selecting a suitable integration strategy is driven by risk assessment of the cloud service provider whose service is subject to continuous audit.

**EXAMPLE**

Let’s assume that the cloud service provider is only willing to subject her cloud service to non-invasive integration of measurement techniques. The reason for this choice is that the risk assessment of the provider has determined that the additional risk entailed with minimally invasive as well as invasive techniques is not tolerable. This implies that presumed benefits of increased transparency provided by continuous security audits are outweighed by the additional risks incurred by configuration and compositional changes.
Let’s be more specific and assume that a provider considers non-invasive integration of a measurement technique to check if communication with his public service endpoints via insecure networks is configured in a secure manner. Through establishing a connection to the endpoint, the desired technique determines if SSL/TLS configuration of a cloud service’s web server allows to securely communicate with the service. To that end, the technique uses a metric to compute a measurement result, that is, a score indicating the strength of the configuration. The underlying model to compute this cipher suite score takes into account known SSL/TLS vulnerabilities such as OpenSSL Heartbleed, CRIME or OpenSSL CCS Injection. Also, the web server must not support TLS fallback signaling cipher suite value (scsv) and secure session renegotiation. Lastly, the web server must not accept self-signed certificates.

The question at this point is: What residual risks does using such a non-invasive measurement technique entail? Let’s first consider the evidence which needs to be produced in order for this technique to calculate measurement results. One example parameter of the technique’s metric is whether the endpoint supports self-signed certificates. Since we are considering a publicly exposed endpoint, this information is public as well, that is, potentially anybody can determine that the endpoint supports self-signed certificates. The same applies to the remaining evidence produced by the measurement technique. This means that anybody may produce the required evidence and compute the cipher suite score. Therefore, one may argue that using this non-invasive measurement technique does not pose any additional risks.

**RISK-DRIVEN INTEGRATION OF EVIDENCE STORE**

Recall that the evidence store is responsible for persisting produced evidence for some predefined period of time. Consequently, the evidence store inherits the challenges of overexposing critical information contained in the evidence as well as protecting evidence against unauthorized alterations (see also Section 2.4.1 of Deliverable 3.3). Therefore, a risk-driven integration of the evidence store is needed, that is, the risk of disclosed, altered or deleted instances of evidence has to be assessed to determine whether the evidence store is integrated as part of the infrastructure of the cloud service under audit or external to the service’s infrastructure, on a remote host. Further, in order to decide how to integrate an evidence store, the additional risk of producing any evidence of any measurement technique using that particular store to persist evidence has to be considered. From the perspective of a cloud provider, the global, additional risk exposure will be determined by the highest additional risk incurred by producing some type of evidence.
EXAMPLE

Recall our example of a non-invasive measurement technique which connects to a cloud service’s endpoints and, based on this evidence, computes a cipher suite score. Here, evidence consists of, e.g., the information that an endpoint possesses some known SSL vulnerability or supports self-signed certificates. As already discussed in the previous paragraph, evidence obtained from this measurement technique is public if the endpoint is publicly reachable.

In context of the integration of the evidence store, this risk exposure is further affected by an evidence store instance which is shared by multiple measurement techniques which are producing evidence for the cloud service. Consider, for example, also storing evidence indicating SQLI vulnerabilities of the cloud service’s web application components. With regard to the evidence store deployment, the question is now – given both types of evidence – what is the global, additional risk exposure? Answering this question, again, depends on the individual risk assessment of the cloud service provider which determines whether to integrate the evidence store as part of the cloud service’s infrastructure or externally.

Note that an evidence store may be shared between multiple cloud service providers, that is, between multiple measurement techniques producing evidence for multiple cloud services and providers. This case can lead to an increase in risk because a successful attack may disclose evidence produced for multiple cloud services of different providers.

RISK-DRIVEN INTEGRATION OF OBJECTIVE EVALUATION APPLICATION

As described in Section 2.4, the objective evaluation application consumes measurement results and, on this basis, reasons about SLOs and SQOs where the outcome of that evaluation is referred to as claims. Both measurement results as well as claims possess a higher level of abstraction than the evidence used to compute the measurement results. Naturally, a strong separation of evidence and measurement results has to be ensured, that is, results forwarded to the objective evaluation application must not contain any evidence used to compute the respective measurement results. Yet, despite a higher level of abstraction, measurement results’ evaluation may still leak information to unauthorized parties, i.e., has a SLO or SQO been satisfied or not. Thus, the additional risk incurred if these results are forwarded to an objective evaluation application not part of the infrastructure of the cloud service under audit has to be assessed.

Note that there may not exist any unauthorized parties if the measurement results and claims are considered to be publicly accessible. In this case, there is no potential damage and thus no
risk to consider when forwarding results from the continuous measurement technique to a remotely hosted objective evaluation application. Otherwise the evaluation application can also be integrated as part of the infrastructure of the cloud service under audit. Note that in the latter case, it is reasonable to expect that the measurement techniques are also integrated in a minimally invasive or invasive manner. Otherwise, evidence produced by the technique as well as computed measurement results exist outside the cloud provider’s infrastructure already.

**EXAMPLE**

Consider, for example, measurement results which indicate whether any persistent storage of the cloud service is encrypted (and only decrypted as needed, e.g., if a query is issued to retrieve some data). To that end, evidence regarding the various types of storage a cloud service may employ, e.g., object storage, relational databases and so forth, has to be produced. This evidence is then provided as input to a suitable metric computing the measurement result at some point in time. This metric may only output a result such as *StorageIsEncrypted* or *StorageIsNotEncrypted*. In this case, it is obvious that if these measurement results were to be disclosed to an unauthorized third party – due to, e.g., vulnerabilities in the objective evaluation application – the potential damage regarding an attacker seeking to cut corners in his attack vector is relatively small since the information obtained is limited.

**RISK-DRIVEN INTEGRATION OF CLAIM STORE**

A *claim* refers to the result of evaluating a control objective stating a control’s satisfaction at a certain point in time. In order to determine whether a control objective is satisfied, one or more measurement results are necessary. A claim is established by the objective evaluation application and then forwarded to the claim store for persistence. The claim store is either part of the infrastructure of the cloud service under continuous audit or hosted on a remote host, external to the service’s infrastructure.

A claim allows deriving what type of measurement result was used to establish the claim. However, it does not tell us anything about the underlying model of the measurement result, that is, the metric which was used to compute the result. Therefore, we cannot directly infer which evidence lead to establishing the claim.

Yet the history of claims may permit conclusions if a control objective is dissatisfied. This, in turn, can translate into time savings on an attacker’s side because the attacker – if a claim’s history is disclosed by unauthorized parties – may filter for potential security issues by absent claims previously satisfied.
The above considerations guide the decision how to integrate a claim store: If disclosing the claim history is considered an intolerable risk, then the claim store can be integrated as part of the cloud service under audit. Note that this does not imply that the objective evaluation application establishing the claims in the first place is also integrated as part of the service’s infrastructure. The reason for this is that the evaluation application does not store any computed claims longer than evaluation requires.

**EXAMPLE**

Consider the claim *During the last 24 hours, the TLS configuration of a cloud service’s endpoint was secure.* Let’s assume this claim has been reissued for some time, e.g., 10 times in succession, suddenly coming to a halt, that is, no such claim is forwarded to the claim store anymore. The absence of such a claim may indicate that the service’s endpoints are not securely configured. This, in turn, can serve as a starting point for an attacker who gained access to the claim store and intends to attack the cloud service under audit.

### 3.2.2 STEP 2: DEPLOYMENT OF TOOL CHAIN

In this step, the continuous security audit tool chain introduced in Section 2.4 is deployed. To that end, it is first necessary to determine the deployment strategy for the tool chain, i.e., where to run certain parts of it. The deployment strategy is derived from the global integration strategy described in the previous section. To that end, each component’s planned integration is inspected and, on this basis, it is determined where to deploy the respective component. Note that although each component of the tool chain can, in principle, be deployed at a different location, it is reasonable to expect that such a fully distributed tool chain is undesired due to various reasons, e.g., performance, reliability and security considerations.

Once deployment of the tool chain is completed, all required components of the tool chain are installed at their desired location. Note that the tool chain is not yet operational since no concrete measurement techniques have been select which, in turn, depends on the components the cloud service consists of. Determining which measurement techniques are feasible and, on this basis, which suitable metrics to select will be described in the next three steps of the integration process.
3.2.3 **STEP 3: DISCOVER CLOUD SERVICE**

In order to determine which measurement techniques can be used in context of a concrete cloud service instance, this step discovers a cloud service including, e.g., applied security policies, components and exposed network services. To that end, complementary discovery techniques are used which assemble available information about a cloud service under audit into a so-called *service description*. A service description, therefore, can be understood as a summary of components and configurations which constitute a particular cloud service.

An instance of a service description is derived from a general model to describe cloud services. An extract of that general model is shown in Figure 3-2 which has been developed on the basis of OpenStack. Depending on the cloud service to be audited, the general cloud service description model is extended, for example, by adding descriptions for specific services provided by Microsoft Azure.

![Figure 3-2: Extract of service description for IaaS provided by OpenStack](image)

It is obvious that the scope of a generated service description depends on the access privileges which discovery techniques are granted by the cloud service provider. These privileges, in turn, result from the global integration strategy selected in the Step 1. Put differently: Discovery techniques are integrated in the same way as are measurement techniques, thus having the same privileges to access components of the cloud service to be audited.

Consider, for example, a cloud service provider only having agreed to a non-invasive integration strategy where, as a consequence, the measurement techniques have to be deployed external to the infrastructure of the cloud service. Given this integration strategy, the discovery techniques also can only discover a cloud service in a non-invasive manner, e.g.,
through scanning its exposed services using tools such as Nmap\textsuperscript{6}. In contrast, when assuming that the provider has agreed to minimally invasive integration, then a discovery technique may be assigned a specific user (e.g., AWS’s security auditor role) with whom it can call admin APIs of the cloud service under audit and retrieve more detailed information about the infrastructure of the cloud service.

**EXAMPLE**

Recall that the selected integration strategy for measurement techniques in our example scenario is non-invasive. Therefore, discovery techniques which can be used to assemble a service description are confined to only interacting with the cloud service’s interfaces, without having privileges to enforce configuration changes (minimally invasive integration) or compositional changes (invasive integration) of the cloud service’s infrastructure.

Let’s assume that as one result of non-invasive discovery, any (publicly reachable) HTTPS endpoint of the cloud service is discovered. More specifically, part of the service description assembled by the discovery techniques contains all publicly reachable IP of hosts which expose port 443, the default port used by HTTPS.

### 3.2.4 STEP 4: DERIVE FEASIBLE MEASUREMENT TECHNIQUES

In this step, feasible measurement techniques are derived by matching the information obtained from the service discovery with the set of all available measurement techniques provided by the tool chain. Thus, feasible measurement techniques denote those techniques which can be actually used in context with a concrete cloud service instance.

In order to identify feasible measurement techniques, the preconditions for each technique have to be identified and modelled as constraints, i.e., a set of rules which has to be satisfied. These preconditions represent assumptions about the environment the technique is operating in as well as the input required by the technique such that it produces complete and correct (i.e., as specified) measurement results. This means that a particular measurement technique can only be used if the cloud service to be continuously audited fulfills the preconditions of the technique.

In the case of test-based measurement using tools such as the Clouditor, the preconditions can be partly derived from the continuous test configuration written in ConTest (see Deliverable 3.2): The input parameters specified for each test cases (the primitive of each continuous test-

\textsuperscript{6} https://nmap.org/
based measurement technique, see Section 4.1.2 of Deliverable 3.2) provide some indication as to what the technique assumes about the environment of the cloud service under audit. As an example, consider having an input parameter `hostname` which suggests that the evidence production technique expects a host which can be reached over an IP-based network. Naturally, assigning semantics to input parameters has to be conducted manually per test case.

Yet inspecting test cases of the test configuration alone does not suffice when eliciting the preconditions of a measurement technique. Additional constraints have to be considered, e.g., the security group granting a remote host – where the technique may be deployed in case of non- or minimally invasive deployment – access the cloud service component to be audited. This is where the service descriptions obtained in the previous step come in: In order to check if such additional constraints are satisfied, additional information about the cloud service have to be available through the service description.

To summarize: In order to derive feasible measurement techniques, the preconditions under which an evidence production technique will work correctly are modelled as a set of rules. These rules draw on the information provided by the service descriptions to check if a particular evidence production technique can be used in context with a concrete cloud service instance.

Lastly, some measurement techniques – even though technically feasible – might not be used at all due to operational risks. This is the case if a technique will foreseeably lead to a significant increase of operational costs of the cloud service infrastructure. Consider, for example, a technique which measures the available bandwidth of a cloud service component where measurement results are used to check whether the available bandwidth is sufficiently high to prevent certain types of Distributed Denial of Service (DDoS) attacks. Furthermore, next to risks originating from increasing operational costs, additional risks can result from the possibility of a measurement technique unintentionally disrupting regular service operation. In this case, the risk consists of a financial loss which is incurred in case of service downtime.

**EXAMPLE**

In order to illustrate the derivation of feasible measurement techniques, consider the following example scenario: Let’s assume that three measurement techniques are available which produce evidence to check which TLS cipher suites an endpoint is using to secure communication via HTTP (i.e., HTTPS). The first technique inspects the configuration used by the webserver which defines the TLS configuration, e.g., accepted cipher suites. The preconditions of this technique require that it has access to the virtual machine where the webserver is running and has sufficient privileges to read the webserver’s configuration file. The second technique obtains the required evidence by connecting to the endpoint and
inspecting what cipher suites are offered by the TLS endpoint at runtime. In order for this technique to work correctly, it has to be able to reach the host exposing the HTTPS endpoint and start a TLS connection, that is, conduct a TLS handshake. The third technique inspects log files generated by the web server and, provided a sufficiently detailed log level, retrieves accepted cipher suites from the log. Similar to the preconditions of the first technique, this technique requires sufficient privileges to access the webserver’s log data. Note that these log files may not only be available at the host where the webserver is running, but also be forwarded to a central logging system using tools such as logstash\(^7\) permitting operational monitoring of a cloud service’s endpoints.

Recall that in the previous section, it was assumed that part of the service description assembled by the discovery techniques contains a publicly reachable IP of a host which exposes port 443. Given this exemplary extract of a service description, it can be concluded that the preconditions of the second evidence production technique are satisfied. This means that the second technique can be used with the example cloud service instance to produce evidence which allows to determine if an endpoint is securing communication via HTTP using strong TLS cipher suites.

### 3.2.5 STEP 5: SELECT SUITABLE METRICS

Having completed Step 4, we now know which specific measurement techniques can be used with a particular cloud service instance. Each measurement technique supports computation of measurement results according to one or more metrics. The question which this step addresses is which measurement results should be produced?

As described in Section 2.4, measurement results serve to evaluate service level objectives (SLO) or service quality objectives (SQO). Yet the problem is that measurement results used to evaluate a SQO or SLO – contrary to their name – cannot be directly measured because they already incorporate an abstraction, i.e., a property model necessary to allow to rigorously evaluate the respective objective. Thus, measurement results are understood as the output of a metric which takes as input the actual raw data, i.e., the evidence which has been obtained by some suitable evidence production technique and, on this basis, performs a predefined computation, thus determining the value of the measurement result.

As laid out in Deliverable 1.4, a SQO is “the commitment a cloud service provider makes for a specific, qualitative characteristic of a cloud service, where the value follows the nominal scale

---

\(^7\) [https://www.elastic.co/products/logstash](https://www.elastic.co/products/logstash)
or ordinal scale (2). Further, a SLO is defined as “the commitment a cloud service provider makes for a specific, quantitative characteristic of a cloud service, where the value follows the interval scale or ratio scale (2). Thus, in order to determine whether a SQO or SLO is satisfied, test metrics have to be available which output measurement results.

- **Measurement results for SQOs**: Characteristics whose values are measured on the nominal scale or ordinal scale imply that reasoning about a SQO is confined to classification and comparison. Put differently: It is at least possible to state whether a cloud service possesses a particular characteristic (nominal level). Consider, as an example, the SQO “User data persisted by the cloud service is encrypted”. Provided having proper measurement techniques available, the value of this characteristic at a certain point in time is either true or false. Further, if a cloud service’s characteristic can be measured on an ordinal level, then measured values can be compared and sorted. For example, a SQO can state that the encryption algorithms used to encrypt sensitive data have to be highly secure. Given a suitable metric, values for this characteristic may be observed indicating insecure, secure and high-secure encryption algorithms where the strict order for these measured values is insecure < secure < high-secure. Intuitively, one may assume that – given the above scale – values observed for secure and high-secure encryption algorithms are somewhat more similar than values indicating insecure and secure algorithms. However, this is incorrect: Measuring on the ordinal scale does not provide any information about the distance between two ranks.

- **Measurement results for SLOs**: Measuring values on the interval as well as on the ratio scale allows to make statements about the difference in measured values. As an example, consider the SLO “A vulnerability of a cloud service has to be fixed within 8 hours after discovery.” Let’s assume that a suitable measurement technique exists which produces the required evidence to compute the desired measurement results allowing to reason about this SLO, e.g., the minutes it took to fix a discovered vulnerability. This measurement result follows an interval scale since the units on the (time) scale are equal to each other, i.e., the difference between 60 and 120 minutes is the same as between 180 and 240 minutes. Further, time is a ratio scale since it possesses a meaningful zero point, thereby permitting comparisons such as fixing the last vulnerability took twice as long as fixing the preceding one.

At this point, it is important to note that it is assumed that a mapping between measurement results and SLOs and SQOs exists which has been agreed upon by domain experts in a prior effort. Having a mapping between measurement results and SLOs and SQOs available means that once feasible measurement techniques have been identified (Step 4), it can be deduced –
based on the feasible metrics these techniques support – which SLOs and SQOs of a concrete cloud service instance can be automatically audited. The selection of suitable metrics from those that are technically feasible then depends on the SLOs and SQOs according to which a cloud service shall be audited continuously.

EXAMPLE

Recall that in Step 4, a feasible measurement technique has been identified which obtains the required evidence by connecting to the endpoint and revealing what cipher suites are offered by the TLS endpoint at runtime. This evidence can serve as input to a set of test metrics which compute measurement results to reason about SLOs and SQOs.

In our example case, this function may inspect the TLS cipher suites offered by the endpoint to check if it only contains suites which are considered strong. These strong cipher suites are predefined in a whitelist, in accordance with the current state of the art. If the endpoint only accepts strong cipher suites, then one feasible metric may output the measurement result is\textit{Strong}. If any other cipher suites are accepted, then the function outputs the measurement result is\textit{NotStrong}. These measurement results follow the nominal scale since they indicate to which group the offered TLS cipher suites belong, that is, either they are all strong (is\textit{Strong}) or they are not all strong (is\textit{NotStrong}).

An example sequence of measurement results obtained by repeatedly executing the evidence measurement technique and computing measurement results by applying the metric may look like this: \textit{<isStrong, isStrong, isNotStrong, isStrong>}. Lastly, having these measurement results available, satisfaction of the following, example SQO can be evaluated: \textit{Every communication channel between the cloud service and a client using HTTP over an insecure network is secured using strong TLS cipher suites}.

Let’s consider another example of a feasible test metric which is based on the measurement technique which obtains TLS cipher suites supported by the cloud service’s endpoints through connecting to them. In this case, measurement results returned by the test metric ought to indicate for how long a cloud service’s endpoint supported one or more cipher suites which are considered insecure, i.e., are not strong. Put differently: The measurement results indicate how long it took the cloud service provider to fix a vulnerable TLS configuration. To that end, the test metric stores the time when it first encounters the cloud service’s endpoint to support TLS cipher suites not considered strong; however, no measurement result is produced just yet. Only the next time when inspecting the evidence indicates that all accepted cipher suites are strong, i.e., the vulnerable configuration has been fixed, a measurement result is produced
whose value follows the ratio scale stating the time (e.g., in seconds) it took to apply the fix. Naturally, this test metric is only feasible if each instance of evidence in this example case contains the time of creation.

An example sequence of measurement results obtained by repeatedly executing the evidence production technique and computing measurement results may look like this: \(<123,345,44,514,78>\). Having these measurement results available, satisfaction of the following, exemplary SLO can be evaluated: *Insecure communication channels which results from misconfigurations have to be fixed within 480 minutes (or 8 hours) after discovering the vulnerable configuration.*

### 3.2.6 STEP 6: START EXECUTION OF MEASUREMENTS

Having selected suitable metrics to reason about SLOs and SQOs, the tool chain is put into operational state by triggering the execution of the measurement techniques required to compute the selected metrics.

### 3.2.7 STEP 7: ADAPT MEASUREMENT TECHNIQUES

Once the initial configuration of the tool chain has been deployed, an additional question is how to adapt to changes in composition as well as in configuration of the cloud service under continuous audit. Such changes may lead to deployed measurement techniques not working correctly anymore, thus not providing correct evidence to compute measurement results. Therefore, it is necessary to continuously check whether the preconditions of deployed measurement techniques are still satisfied. To that end, discovery techniques which are used to assemble service descriptions can be leveraged (see Section 3.2.3). More specifically, these discovery techniques are executed continuously at operation time of the tool chain to check if the information contained in the derived service descriptions still satisfies the set of rules, i.e., the preconditions of a deployed measurement technique.

In case the preconditions of a measurement technique are still satisfied, no further action to adapt the measurement techniques is needed. In case its preconditions are not satisfied anymore, however, this technique is no longer considered feasible. Thus, the operation of the now infeasible technique is terminated. This implies that measurement results which were computed using this evidence cannot be computed anymore and are thus not available to reason about the satisfaction of SLOs or SQOs associated with the measurement results.

Once an infeasible measurement technique has been terminated, the latest service description is then used to find alternative techniques whose outputs, i.e., measurement results are
semantically similar. Note that since the discovery techniques are integrated with the same level of invasiveness as the measurement techniques, it is reasonable to assume that an alternative measurement technique – if existent – is technically feasible.

If such a feasible alternative technique is found, then the remaining question is which risks are incurred by different deployment variants of the alternative technique (similar to Step 2). This means that it is necessary to assess the risks associated with producing evidence and measurement results using the alternative technique. Since evidence instances are used as input to a at least semantically similar test metric, the evidence produced by the alternative technique has to be somewhat similar to the evidence produced by the previously deployed technique. Regarding the information contained in an evidence instance, it can therefore be concluded that evidence produced by the alternative technique is at least as critical as the evidence produced by the previous technique. However, the alternative technique may produce evidence having additional information which increases the associated risk of unauthorized disclosure or alteration. Furthermore, the alternative measurement technique may possess some operational characteristics which increase operational risks as well as costs which should be considered when selecting a deployment variant.

**EXAMPLE**

Recall the example SQO *Every communication channel between the cloud service and a client using HTTP over an insecure network is secured using strong TLS cipher suites*. Let’s assume that the cloud service under audit has changed in the following way: As result of increased security needs of the cloud provider, previously publicly reachable endpoints are now confined to only a few whitelisted hosts. Therefore, the non-invasive measurement technique which checked the supported TLS cipher suites by connecting to the endpoints is not feasible anymore. However, the cloud provider has exposed an existing Audit API which centrally exposes information about supported TLS suites of any of his service endpoints to authorized parties. Therefore, an alternative technique may call the Audit API to produce evidence and measurement results which are semantically similar to those results produced by the previous technique.
4 TECHNICAL INTEGRATION WITH CLOUD SERVICES

This chapter describes how the previously mentioned tools can integrate with existing cloud services, in order to gather information for continuous auditing.

4.1 APPLICATION LEVEL INTEGRATION

The goal of this section is to show how the tool chain described in Section 2 can be integrated with SaaS applications on the application level. Application level integration allows to produce application level evidence and measurement results. This, in turn, permits to evaluate control objectives on the application level.

To enable the tool chain to continuously audit cloud services on the application level, a measurement techniques’ implementation such as Clouditor needs to be able to access a given API. In the following, we describe how the EU-SEC Continuous Audit API (CA API) can be used to achieve this. The design of the CA API was driven by the requirements defined in Task 5.1 of Working Package 5 which is responsible for preparing the contiguous auditing pilot. The goal of this API is to be as agnostic as possible by basing its design on industrial standards. However, at certain points (environment, unique identifier structure, example calls) the EU-SEC CA API becomes application-specific. The definition of the CA API itself can be found in Deliverable 5.1, Chapter 6. Furthermore, the specification of the CA API itself was made open-source and can be found on GitHub⁸, including example clients how to access the API in Java⁹.

In the following, to illustrate the integration process on a practical example, the Fabasoft Cloud, which is used in one of the pilot variants in WP5, is used.

4.1.1 ENVIRONMENTS

Modern cloud services often provide multiple environments, to support different phases of the development process, such as testing, staging and production.

⁸ [https://github.com/eu-sec/continuous-auditing-api-spec](https://github.com/eu-sec/continuous-auditing-api-spec)
⁹ [https://github.com/eu-sec/continuous-auditing-api-java-client](https://github.com/eu-sec/continuous-auditing-api-java-client)
Fabasoft follows this example by providing multiple environments for development/testing and for production usage. The environments for development/testing are called Fabasoft VDE (Virtual Development Environment). The production environment is the Fabasoft Cloud. For the purposes of the EU-SEC project, Fabasoft provided a dedicated VDE (dev4/vm114), which will be referred to as Fabasoft VDE for the rest of the section.

Fabasoft currently operates three data locations (governance regions). Each data location is addressed by a specific URL, one for Austria, one for Germany and one for Switzerland. The physical locations for these data locations are documented in the “Performance Characteristics Data Centers”.

The user accounts for customers of the Fabasoft Cloud are entirely managed by the customer, either by registering an account via https://www.fabasoft.com/register, by ordering a dedicated tenant via cloudsales@fabasoft.com or by invitation of an existing user in the Fabasoft Cloud.

**4.1.2 ACCESSING AUDIT DATA**

The Fabasoft VDE/Fabasoft Cloud provides access for continuous auditing by existing protocols and by new, dedicated EU-SEC CA API web service calls, developed in this project.

The following base URLs must be used in order to access information in the Fabasoft VDE/Fabasoft Cloud:

  - For testing & development purposes, e.g., Pilot 2 in Working Package 5
- **Fabasoft Cloud**
  - Data location "Austria": [https://at.cloud.fabasoft.com/folio](https://at.cloud.fabasoft.com/folio)
  - Data location "Germany": [https://de.cloud.fabasoft.com/folio](https://de.cloud.fabasoft.com/folio)
  - Data location "Switzerland": [https://ch.cloud.fabasoft.com/folio](https://ch.cloud.fabasoft.com/folio)

These base URLs are valid for both existing protocols and for dedicated EU-SEC CA API web service calls.

**ACCESSING OBJECT INFORMATION**

The EU-SEC CA API provides details on an object-level, meaning that it retrieves a certain type of information, for example encryption details on a particular object under a certain scope. The

---

10 see https://www.fabasoft.com/data-center
following code shows the API definition for said call according to the OpenAPI/Swagger standard.

`/{scope}/persistence/{objectId}/encryption`:

get:
  tags:
    - ca_api_persistence
  summary: Retrieves the encryption info of an object.
  description: >-
    Based on CCM-EKM-04. Retrieves the encryption info of an object. Proper interpretation has to be performed by the audit tool.
  operationId: getEncryptionInfo
  produces:
    - application/json
  parameters:
    - name: objectId
      in: path
      description: ID of either objectId on SaaS level or storageId on lower level
      required: true
      type: string
    - name: scope
      in: path
      description: Scope of the service
      required: true
      type: string
  responses:
    '200':
      description: successful operation
      schema:
        $ref: '#/definitions/EncryptionStorageResponse'
    '405':
      description: Invalid input

For the Fabasoft integration, only the scope `application` is used. The format of the objectID is specific to the application. For example, in the Fabasoft VDE/Fabasoft Cloud each Object is identified by a unique identifier. The identifier has the format "COO.a.b.c.d".

The Fabasoft VDE/Fabasoft Cloud provides two standard protocols to retrieve a list of all object identifiers in the repository:

- CMIS (Content Management Interoperability Services)<sup>11</sup>
- WebDAV (Web-based Distributed Authoring and Versioning)<sup>12</sup>

The curl command line

```
curl -X GET -u kimble0001:PASSWORD "https://vde.fabasoft.com/dev4/vm114/folio/cmis"
```

<sup>11</sup> see https://en.wikipedia.org/wiki/Content_Management_Interoperability_Services, or http://docs.oasis-open.org/cmis/CMIS/v1.1/CMIS-v1.1.html

will provide the following XML data – the highlighted line will provide the URL to the children of the root element of user kimble0001\textsuperscript{13}:

```
<?xml version="1.0" encoding="UTF-8"?>
[-]
<title>FscDucx</title>
</app:collection>
[-]
<cmis:collectionType>root</cmis:collectionType>
</app:collection>
[-]
<cmis:collectionType>templates</cmis:collectionType>
</app:collection>
<cmis:repositoryInfo>
<cmis:repositoryId>COO.200.200.1.1975</cmis:repositoryId>
<cmis:repositoryName>FscDucx</cmis:repositoryName>
<cmis:repositoryDescription></cmis:repositoryDescription>
[-]
</cmis:uritemplate>
</app:workspace>
</app:service>
```

The curl command line

```
```

will provide access to the children of the root element of userkimble0001 (with id COO.200.200.1.1975) and so on.

Web Services to the Fabasoft VDE are authenticated via Basic Authentication, so the https requests of the web services must contain the basic authentication credentials (username and password).

Web Services to the Fabasoft Cloud are authenticated via Basic Authentication, but the password provided is a “Password for Application” configured in the account menu of the user, that wants to allow web service access.\textsuperscript{14}

Once all relevant object identifiers are known, the continuous audit tool can the access audit relevant data through the appropriate EU-SEC CA API Call, for example:

\textsuperscript{13} More information about the Fabasoft CMIS implementation can be found here: https://help.folio.fabasoft.com/doc/Fabasoft-Integration-for-CMIS/index.htm

4.2 PLATFORM LEVEL INTEGRATION

The goal of this section is to show how the tool chain described in Section 2 can be integrated with IaaS on the platform level (i.e., integration with IaaS control plane). Platform level integration allows to produce platform level evidence and measurement results. This, in turn, permits to evaluate control objectives on the platform level. In the following section, we describe a selection of IaaS provided Amazon Web Services (AWS).

4.2.1 ENVIRONMENT

Amazon Web Services (AWS) follows the so-called shared responsibility model\(^\text{15}\) which denotes that the responsibility to operate a cloud service secure is shared between the customer and AWS as a cloud provider: while AWS makes sure that its services are not vulnerable to attacks, customers have to configure AWS services which they use in a secure manner. This means that AWS takes no responsibility for, e.g., incorrectly configured customer security groups or vulnerable applications the customer may choose to deploy.

In this context, platform-level (or control plane) integration in the case of AWS delineates the integration of continuous auditing tool chain with an AWS customer. It does not mean, however, that the tool chain integrates with the underlying cloud infrastructure directly maintained by AWS.

The current service portfolio of AWS consists of more than 100 services. With regard to the continuous auditing pilot of Working Package 5, the following four services are considered herein:

\(^{15}\) https://aws.amazon.com/compliance/shared-responsibility-model/
• Amazon Elastic Compute Cloud (EC2): Computing resource service.
• Amazon Elastic Block Storage (EBS): Volumes for EC2 instances.
• Amazon Simple Storage Service (S3): Object storage.
• Amazon Rational Database Service (RDS): Managed rational database service supporting, e.g., MySQL.
• AWS Key Management Service (KMS): Encryption and management of cryptographic keys.

4.2.2 PLATFORM APIS

Configuration information about EC2, EBS, S3, RDS and KMS required to determine whether control objectives are met on the platform level can be retrieved using the AWS API of the respective service. APIs are supplied as part of AWS SDKs which are available for multiple languages.¹⁶

4.2.3 EXAMPLE TEST-BASED MEASUREMENTS

Different to the example integration on application level where Fabasoft provided a dedicated Audit API to support control objective checks, AWS does not (yet) offer such an API on the platform level. Therefore, we have to draw on the AWS APIs to design the test-based measurement techniques outlined hereafter. These example techniques are selected based on the identified requirement provided by Task 5.1 of Working Package 5.

• Location of S3 objects: Determine location of data stored in S3 buckets.
• Encryption status of objects stored in S3 buckets: Determine if all objects stored in S3 are encrypted.
• Default encryption of object storage (bucket level): Determine if default encryption for an S3 bucket is enabled.
• S3 Encryption policy (bucket level): Determine if any S3 Bucket has an encryption policy
• Encryption of EBS volumes: Determine if all EBS volume are encrypted.
• Encryption status of databases provided by RDS: Determines all DB instances are encrypted.
• Origin of KMS keys: Determine if the KMS keys have the correct origin (expected: 'external').

¹⁶ https://aws.amazon.com/tools/
• Key rotation of KMS keys: KMS keys have key rotation enabled (only applicable to non-external keys).
5 EVALUATION OF CONTINUOUS TEST-BASED MEASUREMENT TECHNIQUES

As pointed out in the Introduction of this document, erroneous test results can decrease customers' trust in test results and can lead to providers disputing results of a continuous test-based security audits. In order to address this challenge, this chapter introduces a method how to experimentally evaluate the accuracy and precision of continuous test-based measurement techniques. This method allows to compare alternative test-based measurement techniques as well as compare alternative configurations of test-based techniques. Furthermore, it permits to infer general conclusions about the accuracy of a specific test-based measurement technique. Parts of the contents of this chapter have been published in (4), (5) and (6).

The next section introduces four universal metrics which can be used with any test-based measurement technique and, on this basis, defines the terms accuracy and precision in the context of such test-based techniques. Thereafter, Section 5.2 provides a high-level overview of how the method works and Section 5.3 describes how to violate of cloud service properties leading to dissatisfaction of SLOs or SQOs and thus non-compliance of the service with a certificate's controls. Then Section 5.4 introduces accuracy and precision measures applicable to any test-based measurement technique, including the inference of conclusions about the general accuracy of a test-based technique. Finally, Section 5.5 presents experimental results of applying our method to evaluate and compare exemplary continuous test-based measurement techniques which aim to support certification of controls related to property secure communication configuration.

5.1 BACKGROUND

In this section, first four universal test metrics are presented which can be used with any test-based measurement technique which strictly follows the building blocks described in Section 4.1 of Deliverable 3.2. Thereafter, Section 5.1.2 introduces basic measuring as well as statistical terminology and concepts which are required for experimental evaluation.
5.1.1 UNIVERSAL METRICS FOR TEST-BASED MEASUREMENT TECHNIQUES

Test-based measurement techniques seek to automatically and repeatedly produce measurement results which allow to check if a cloud service satisfies a set of objectives (i.e., SLOs and SQOs) over time. Recall that metrics take as input evidence provided by test-based techniques and output measurement results. Measurement results, in turn, are used to reason about SLOs and SQOs. Continuous test-based measurement therefore implies that a sequence of instances of evidence have to be interpreted by suitable metrics in order to produce measurement results which, in turn, allow to reason about defined objectives over a period of time.

Recall that in Section 4.1 of Deliverable 3.2, the building blocks of test-based measurement techniques to continuously produce measurement results to be used for security audits were presented. Test cases form the primitive of each continuous test which use test oracles to determine the outcome of a test case, that is, whether a test cases passes or fails. Further, test suites combine test cases where each suite contains at least one test case. A test suite either passes or fails, it passes if all contained test cases pass.

Note that the definition of metric used here refines the one provided by Deliverable 1.4: We describe a metric as a function $M: R \rightarrow U$ which takes as input results of test suite runs $R$ and outputs measurement results $U$. A metric can be computed based on any information available from the result of a test suite run, e.g., at what time the test suite run was triggered, when it finished, and further information contained in the results of test case runs bound to the test suite run.

Any test metric used by a test-based measurement technique which strictly follows the building blocks defined in Section 4.1 of Deliverable 3.2 can therefore make use of the following two characteristics: First, a single test suite run (i.e., a single execution of a test suite as part of a continuous test) either passes or fails. As a consequence, and second, a single test suite run passes or fails at some point in time. Based on these two key characteristics, four test metrics functions are proposed hereafter which are universally applicable to any type of evidence.

**BASIC-RESULT-COUNTER (BRC)**

A basic test result $br$ tells us if a test failed (f) or passed (p), i.e., $br \in \{f, p\}$. The Basic-Result-Counter ($brC$) metric takes any instance of $br$ as input and counts the number of times a test failed ($brC^f$) or passed ($brC^p$).
As Figure 5-2 shows, a basic test result is only returned after the execution of a test suite run completed \((\text{tsr}_i)\). This metric can be used to assess statements only requiring to evaluate if and how often a continuous test failed or passed. Consider, as an example application, determining if and how often security groups assigned to a newly started virtual machine unexpectedly allow that these machines are publicly accessible through other than whitelisted ports.

**FAILED-PASSED-SEQUENCE-COUNTER (FPSC)**

A continuous test repeatedly produces basic test results. A failed-passed-sequence \((fps)\) is a special sequence of basic test results: As Figure 5-1 shows, a \(fps\) starts with a failed test at \(t_i\) given that the previous test at \(t_{i-1}\) passed. An \(fps\) ends with next occurrence of a passed test.

\[ fps = \langle f, f, f, f, f, f, p, p, p \rangle \]

**Figure 5-1 Exemplary failed-passed-sequence \((fps)\) based on basic test results \((br)\)**

For example, consider having observed the following sequence of basic test results produced by a continuous test: When attempting to connect to a VM for eleven times in a row, the first two times the login were successful \((p)\). However, for the next six times, the login fails \((f)\) and for the remaining three times, the test succeeds again. The example \(fps\) is the sequence \(fps_{sch}^{11} = \langle f, f, f, f, f, f, p, p, p \rangle\).

The Fail-Pass-Sequence-Counter \((fpsC)\) metric uses this definition of \(fps\). \(fpsC\) counts the number of occurrences of \(fps\) which are observed within a sequence of basic test results \(S_{br} = \langle br_1, br_2, \ldots, br_i \rangle\) produced during a continuous test. Consider, as an example, Figure 5-1 which shows the following sequence of basic test results \(S_{br} = \langle p, p, f, f, f, f, f, p, p, p \rangle\). Sequence \(S_{br}\) contains exactly one \(fps\), i.e., \(fpsC(S_{br}) = 1\).

**FAILED-PASSED-SEQUENCE-DURATION (FPSD)**

The Fail-Passed-Sequence-Duration \((fpsD)\) metric draws on the definition of a failed-passed-sequence \((fps)\). \(fpsD\) takes a \(fps\) as input and measures the time between the first failed test of an \(fps\) and its last basic test result which passes by definition. This test metric allows to reason about properties over individual periods of time, thus it can be used to evaluate
statements which contain time constraints. Consider, for example, a control implementation derived from, e.g., RB-21: Handling of vulnerabilities, malfunctions and errors – check of open vulnerabilities of BSI C5 (7) that an incorrectly configured and thus insecure webserver’s TLS setup of a SaaS application is fixed within a certain amount of time, e.g., eight hours.

The definition of \( fpsD \) has a subtle detail: Recall that \( fpsD \) aims to measure the time difference between the first and the last test of a failed-passed-sequence, that is,

\[
fps = (f_i, f_{i+1}, f_{i+2}, ..., p_{i+j}).
\]

It is important to note at this point is that the first failed test \( f_i \) as well as the next passed test \( p_{i+j} \) each have a duration themselves. This means that both tests take some time to complete and return a basic test result. As a consequence, we have to select whether a \( fpsD \) starts at the start time or the end time of the first test \( f_i \). Further, we have to decide whether a \( fpsD \) ends at the start time or the end time of the last test \( p_{i+j} \).

In order to properly define the limits of a \( fpsD \), we have to first shed light on different options which may affect our metric. For example: Figure 5-2 illustrates the definition of \( fpsD \) which uses the start time \( tsr_i^e \) of the first failed test \( tsr_i \) and the end time \( tsr_i^e \) of the next passed test \( tsr_{i+j}^e \). Note that duration of the first failed test is \( d_i \) and duration of the last passed test is \( d_{i+j} \).

![Figure 5-2 Example definition for universal test metric fpsD](image)

It is obvious that the example definition of \( fpsD \) shown in Figure 5-2 has a downside: The more time it takes the last test \( tsr_{i+j}^e \) to complete, the higher the proportion of \( d_{i+j} \) within the \( fpsD \). Therefore, choosing \( tsr_i^e \) and \( tsr_{i+j}^e \) as bounds for \( fpsD \) makes \( fpsD \) dependent on the duration of \( tsr_{i+j} \). For scenarios requiring high accuracy of \( fpsD \), e.g., to evaluate statements defining narrow time constraints, this dependency can make the metric \( fpsD \) unsuited.

As already pointed out in the introduction of this section, the metric \( fpsD \) ought to be applicable to any continuous test. This means that a definition of \( fpsD \) has to avoid dependencies of the duration of a specific last test \( tsr_{i+j} \). In order to derive a definition of \( fpsD \)
least dependent on test suite runs’ duration, we have to analyze how variations in the duration of the first failed test \( d_i \) and the last passed test \( d_{i+j} \) impact on \( fpsD \).

Figure 5-3 shows the four available options to define \( fpsD \). Let us consider, for example, Option 3: Here, the end of the first failed test \( tsr_i^e \) is used as start of the \( fpsD \) while the end of the next passing test \( tsr_{i+j}^s \) serves as end of the \( fpsD \). When selecting this definition, variations of either the duration of the first test \( \Delta d_i \) as well as the last test \( \Delta d_{i+j} \) will impact on \( fpsD \), i.e., result in \( \Delta fpsD \). Also, variations of both tests \( \Delta d_i \wedge \Delta d_{i+j} \) also change \( fpsD \), i.e., \( \Delta fpsD \). Note that there exists a corner case where duration variations of both tests cancel each other out, that is, if \( \Delta d_i = d_{i+j} \), then \( fpsD \) remains unaffected.

<table>
<thead>
<tr>
<th>Option</th>
<th>Start time</th>
<th>End time</th>
<th>( \Delta d_i )</th>
<th>( \Delta d_{i+j} )</th>
<th>( \Delta d_i \wedge \Delta d_{i+j} )</th>
<th>( \Delta d_i \neq \Delta d_{i+j} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>( tsr_i^e )</td>
<td>( tsr_{i+j}^s )</td>
<td>( fpsD )</td>
<td>( \Delta fpsD )</td>
<td>( \Delta fpsD )</td>
<td>( \Delta fpsD )</td>
</tr>
<tr>
<td>2</td>
<td>( tsr_i^e )</td>
<td>( tsr_{i+j}^s )</td>
<td>( fpsD )</td>
<td>( fpsD )</td>
<td>( fpsD )</td>
<td>( fpsD )</td>
</tr>
<tr>
<td>3</td>
<td>( tsr_i^s )</td>
<td>( tsr_{i+j}^e )</td>
<td>( \Delta fpsD )</td>
<td>( \Delta fpsD )</td>
<td>( \Delta fpsD )</td>
<td>( \Delta fpsD )</td>
</tr>
<tr>
<td>4</td>
<td>( tsr_i^s )</td>
<td>( tsr_{i+j}^e )</td>
<td>( fps)</td>
<td>( fps )</td>
<td>( fps )</td>
<td>( fps )</td>
</tr>
</tbody>
</table>

**Figure 5-3 Available options to define Fail-Pass-Sequence-Duration (fpsD) if \(|fps| > 2\)**

When inspecting Figure 2-1, it is obvious that Option 2 is the only definition of \( fpsD \) unaffected by variations of duration of the first and the last test suite run. Therefore, we define the start of a \( fpsD \) to be the start time of the first failed test (i.e., \( tsr_i^e \)) while the end of a \( fpsD \) is the start time of the next passed test (i.e., \( tsr_{i+j}^s \)).

Note that the reasoning shown in Figure 5-3 is only true if the failed-passed-sequences contains more than two basic test results, that is, \(|fps| > 2\). In case a fps only containing two elements, i.e., \( fps = \langle f_i, p_{i+1} \rangle \), then variations of the duration of the failing test \( f_i \) will lead to changes of \( fpsD \). Furthermore, if \(|fps| = 2\), then the duration of the \( fpsD \) will be at least as long as it takes the failing test to complete. Consequently, the time it takes to complete the first failing test also defines the lower bound on how accurately we can reason about statement containing time constraints.

**CUMULATIVE-FAILED-PASSED-SEQUENCE-DURATION (CFPSD)**

This metric builds on the failed-passed-sequence-duration (fpsD) presented in the previous paragraph. The input to test metric \( cfpsD \) is a sequence \( \hat{fpsD} \) consisting of any \( fpsD \) observed during a continuous test, and, on this basis, \( cfpsD \) outputs their accumulated value.
The metric $\text{cfpsD}$ allows us to reason about cloud service properties within a predefined period of time. Similar to the metric $\text{fpsD}$, we can leverage $\text{cfpsD}$ to evaluate statements containing time constraints. Different to $\text{fpsD}$, however, $\text{cfpsD}$ permits to evaluate statements whose time constraints refer to multiple property violation events observed within a particular period of time. As an example, consider a service level agreement which defines that the total yearly downtime of a cloud service must not surpass five minutes (Note that compliance with SLAs is required by various controls, e.g., RB-02 Capacity management – monitoring of the Cloud Computing Compliance Controls Catalogue (BSI C5) (7)). During the period of a year, the cloud service exhibits multiple, timely separated downtime events which are detected by a suitable continuous test. The metric $\text{fpsD}$ can be used to evaluate statements which contain a single downtime event to, e.g., not last longer than 60 seconds. In contrast, $\text{cfpsD}$ takes a period of time into account, e.g., a year, and summarizes over any $\text{fpsD}$ observed to evaluate statements that refer to all downtime events during the entire period.

5.1.2 ACCURACY AND PRECISION

In the previous section, four universal test metrics for test-based measurement techniques have been introduced which allow us to evaluate SLOs and SQOs defined for cloud services. The question at this point is: What errors do these measurements results possess and how do these errors affect our conclusion about whether a cloud service satisfies a SLO or SQO.

In this section, it is will defined what accuracy and precision mean in the context of measurement results produced by the four universal test metrics. To that end, we draw on standard measurement theory and statistical methods used within various fields of experimental science. The basic definitions of concepts such as accuracy and precision used within this section follow (8), (9) and (10). Furthermore, statistical methods leveraged within this section are comprehensively covered in the literature, e.g., (11) (12) (13).

ACCURACY

The accuracy of the measurement describes whether the measured value agrees with the accepted value. This accepted or true value can be provided by previous observations or theoretical calculations. The concept of accuracy thus only applies if experimental data is analyzed with the goal to compare the experimental results with known values.

Recall the four test metrics $\text{brC}$, $\text{fpsC}$, $\text{fpsD}$ and $\text{cfpsD}$ which have been introduced in Section 5.1.1. The accuracy of measurement results produced by these test metrics are outlined hereafter:
- **Basic-result-Counter (brC):** This metrics counts the number of passed and failed tests. A basic test result is accurate if it indicates that a control is not satisfied by the cloud services at a time where the cloud service indeed does not comply with the control. Also, a basic test result is accurate if it indicates satisfaction of a control by a cloud service at a time where the service indeed complies with the control.

- **Failed-Passed-Sequence-Counter (fpsC):** This metrics counts the number of observed failed-passed-sequences (fps). A fps is accurate if the cloud service actually does not comply with a control during the time indicated by the fps.

- **Failed-Passed-Sequence-Duration (fpsD):** This metric describes the time elapsed between the first failed test and the last passed test of a fps. A measurement result produced by fpsD is accurate if it agrees with the actual duration of temporary non-compliance of a cloud service.

- **Cumulative-Failed-Passed-Sequence-Duration (cfpsD):** This metric describes the accumulated time during which a control is not satisfied. A measurement result produced by cfpsD is accurate if it matches the actual duration of the temporary non-compliance of cloud service within a specified interval.

The reason why measured values may not agree with accepted values are **systematic errors.** These errors may result from, e.g., erroneous implementation and configuration of the measuring device. Identifying the causes of systematic errors is usually non-trivial where, in the case a test-based measurement technique, this measuring device consists of any component used to implement the test-based measurement technique, that is, any component implementing the framework to design continuous test described in Chapter 4 of Deliverable 3.2.

Systematic errors of measurement results vary depending on the test metric. In Section 5.4, accuracy measures for each of the four universal test metrics will be explored which allows to quantify the disagreement between measured values and true values.

Furthermore, as will be detailed in Section 5.3, true values are established through intentionally manipulating cloud services to not satisfy a SLO or SQO which measurement results produced by the universal test metrics aim to check. Thus, we know the true values and can compare them with the measured ones provided by the evidence production technique under evaluation, thereby providing us with the accuracy of the technique. However, the remaining problem is that the systematic error measurement results may exhibit can vary due to **random errors.** This brings us to the concept of precision which is explained in the following section.

**PRECISION**
**Precision** refers to the closeness of agreement between successively measured values conducted under identical conditions (9). When neglecting systematic errors, then those repeatedly executed measurements provide a range of values spreading about the true value. The reason for this spread are *random errors* which are caused by unknown and unforeseeable changes in the experiment, e.g., fluctuation in the network delay due to electronic noise. The smaller the random errors, the smaller the range of values, and thus the more precise the measurement (8). Hence, the level of precision of experimental measurements is determined by random errors.

- **Arithmetic mean:** Assume having observed some repeated measurements \( X = \langle x_1, x_2, \ldots, x_n \rangle \) only having random errors. The question now is: What is true value of these measurements? In statistical terms, the answer is to use the values of sample distribution \( X \) to estimate the expected value \( \mu \) of the parent distribution \( Y \). The best estimate for \( \mu \) to be derived from these measurements is the arithmetic mean. Using the values of \( X \), we compute the sample mean

\[
\bar{x} = \frac{1}{n} \sum_{i=0}^{n} x_i
\]

serving as our estimate of \( \mu \). Averaging follows the intuition that random errors are equally likely to be above as well as below the true value. Thus, averaging evenly divide the random error among all observations.

A special case arises if the values of \( X \) and \( Y \) can only assume one of two values, for example, 0 or 1. In this case, computing the arithmetic mean give us the fraction of values with 1’s of \( X \). This is referred to as the sample *proportion* \( \bar{p} \) which serves as an estimate of the population proportion \( p \).

At this point, it is important to note that the assumption of our measurements in \( X \) only having random errors is rather theoretical. In a real experiment, each \( x \in X \) will possess random errors and systematic errors. Therefore, \( \bar{x} \) or \( \bar{p} \) are *not* estimates for their true value, they provide estimates for their true values *plus* their systematic errors.

Estimating the population mean \( \mu \) and population proportion \( p \) based on \( \bar{x} \) and \( \bar{p} \) works because of the laws of large numbers: The *weak law of large numbers* states that if the number of samples \( n \) generated from the distribution \( Y \) goes to infinity, then the probability of making a random error larger than \( \epsilon \) goes to zero:

\[
\lim_{n \to \infty} P(|\bar{x}_n - \mu| > \epsilon) = 0.
\]
Furthermore, the strong law of large numbers states that the probability of the sample mean $\bar{x}_n$ converging to the expected value is 1:

$$P\left(\lim_{n \to \infty} |\bar{x}_n - \mu| = 0\right) = 1.$$  

Both laws of large numbers suggest that provided a sufficiently large number of samples, – i.e., take a sufficient large number of measurements – we can produce an estimate $\bar{x}$ with a random error $\epsilon = |\bar{x}_n - \mu|$ which can be as small as we desire. Put differently: Given a sufficiently large number of measurements, the estimate converges to the true value plus systematic error. Yet neither law tells us how many measurements have to be conducted to reduce $\epsilon$ below a particular threshold.

- **Standard deviation**: The sample mean $\bar{x}$ estimates the true value plus systematic errors. However, it does not provide us with any information on the range of measured values. To describe the width of the sample distribution $X$, we can use the standard deviation $s_d$:

$$s_d = \sqrt{\frac{1}{|X|} \left( (x_1 - \bar{x})^2 + (x_2 - \bar{x})^2 + \cdots + (x_i - \bar{x})^2 \right)}.$$  

The standard deviation considers any values of $X$ and provides the average distance of a measurement value to the mean. If we observe another measurement and want to know if it is a common or exceptional value, then we can make use of $s_d$. First, we standardize the observed value $x$ by computing z-scores:

$$z = \frac{x - \bar{x}}{s_d}.$$  

Whether a $z$ value is low or high depends on the distribution of $X$: In case of a normal distribution, 99% of the values lie within z-scores of [-3,3] where any value outside this range may be considered exceptional.

The $s_d$ has one important disadvantage: Adding more measurement values to $X$ increases the precision with which we can estimate the population mean $\mu$ since it decreases the random error. Yet when conducting more measurements, the standard deviation of $X$ remains relatively stable. This means that the standard deviation is not a good measure to describe the error of the sample mean, that is, the closeness of the sample mean to the population mean.

- **Standard error**: Having estimated the population mean $\mu$ with $\bar{x}$, the standard error $se$ is the suitable choice when intending to describe the precision of $\bar{x}$. The $se$ is the standard deviation of the so-called sampling distribution. Note that we have already
seen two distributions, that is, the parent distribution $Y$ whose expected value we aim to estimate using sample distribution $X$ which contains the samples drawn from $Y$. The sampling distribution is a theoretical distribution which were to obtain if we draw all possible samples $X$ from $Y$ and compute a statistic, e.g., the mean of each of these samples. Naturally, in practice, this is usually impossible or not desired. The resulting distribution of all these means is the sampling distribution of the mean.

The calculation of the standard error depends on the statistic. The se for the sample mean $\bar{x}$ can be obtained as follows:

$$se_{\bar{x}} = \frac{sd_{\bar{x}}}{\sqrt{n}}$$

It is obvious that an increasing standard deviation $sd$ of the sample distribution $X$ leads to a higher standard error. However, the standard error decreases if the number of samples in $X$, that is, $n$ increases.

Further, the standard error for a sample proportion $\bar{p}$ is computed as follows:

$$se_{\bar{p}} = \sqrt{\bar{p} \times \left(1 - \bar{p}\right) \over n}.$$  

- **Confidence intervals**: Combining the notion of the standard error with the assumption that the sampling distribution approximately follows a normal distribution permits estimating the precision of the sample mean and the sample proportion by constructing confidence intervals for the sample mean and for the sample proportion. In contrast to point estimation like $\bar{x}$ and $\bar{p}$, confidence intervals are a special type of interval estimates which give a range of probable values of an unknown parent's distribution parameter.

In order to construct a confidence interval, it is necessary to decide on a confidence level and then compute the desired statistic, e.g., sample mean $\bar{x}$, as well as the margin of error ($E$).

- **Confidence level (CL)**: The fraction of all possible samples expected to include the true parameter of the unknown parent distribution. Consider, as an example, all possible samples $X$ are drawn from the distribution of $Y$ and for each a 99% confidence interval for the sample mean is computed. In this case, 99% of the computed confidence intervals would include the population mean, i.e., the mean of the distribution of $Y$. 
• **Statistic:** The property of a sample which is used to estimate population parameter’s value. In our case, we use the sample mean $\bar{x}$ and the sample proportion $\bar{p}$.

• **Margin of error ($E$):** This margin defines the interval estimation by the the range above and below the sample statistic. The calculation of $E$ depends on the standard error which, in turn, depends on the selected statistic. For the sample mean $\bar{x}$, the margin of error is

$$E_x = t_{CL} \times se_x.$$  

$t_{CL}$ is the value that separates the middle the area of the $t$-Distribution according to the selected confidence level $CL$, e.g., 95%, and the standard error of the mean $se_x$.

For the sample proportion $\bar{p}$, the margin of error is

$$E_p = z_{CL} \times se_p.$$  

$z_{CL}$ is the z-value that separates the middle area of the standard normal distribution according to the chosen confidence level $CL$, e.g., 99%, and the standard error for the proportion $se_p$.

• **Calibrating precision:** Recall that at the end of paragraph on the arithmetic mean, it was described that the laws of large numbers justify making a point estimate of a parent’s distribution parameter, e.g., using the sample mean $\bar{x}_n$ to estimate the mean $\mu$ of the distribution of $Y$. Yet we do not know how close this estimate is to the true value (plus systematic error), that is, how large is the error $\epsilon$ for a given sample size $n$?

After introducing confidence intervals’ construction for sample means and proportions, we can now leverage the following idea: The sample size $n$ can be used as a parameter to determine the number of samples needed to achieve a desired margin of error $\hat{E}$, that is, the desired precision. To that end, $\hat{E}_p$ and $\hat{E}_\bar{x}$ are solved for the sample size $n$ which gives us

$$\hat{n}_p = \frac{z_{CL} \times \bar{p} \times (1 - \bar{p})}{\hat{E}^2}$$  

and

$$\hat{n}_\bar{x} = \frac{sd_x \times t^2_{CL}}{\hat{E}^2}.$$  

In practice, one apparent problem of solving these formulas is that they have to be solved prior to executing the experiment to evaluate a test-based measurement technique. This means that there may not exist any previously observed values to plug in for $\bar{p}$ and $sd_x$. This would leave us with an educated guess of these values, otherwise we may use historical values previously observed.
5.2 OVERVIEW OF THE EVALUATION PROCESS

The accuracy and precision of measurement results produced by a specific test-based measurement technique depend on various factors, such as implementation of the test, test environment and usage of external tools. Without experimental evaluation, it is thus hard to make a statement on how well a test-based measurement technique works in detecting a control’s (i.e., a SLO’s or SQO’s) satisfaction or violation.

The approach described hereafter treats a test-based measurement technique under evaluation as a black box. Therefore, no information about the internal composition and implementation of the technique is needed, e.g., if and which external tools are used. Only measurement results produced by the test-based measurement technique during an experiment are observed where the violations of the control, that is, violations of the SLOs or SQOs associated with the control are induced which the technique intends to validate. Put differently: Correct results as well as errors of the test-based technique under evaluation follow some unknown distributions. Samples from these unknown distributions are taken by running experiments where controls are intentionally violated. Based on these experiment results, conclusions about the accuracy of the test-based measurement technique are drawn.

Figure 5-4 provides a high-level overview of our method. As part of configuring a control violation sequence, duration of and time between each control violation event is randomized within some specified limits (Step 1). Then the test-based technique is configured according to the building blocks described in Deliverable T3.2, Section 4 (Step 2): Selecting test cases, setting test suites parameter and choosing a workflow. Thereafter, the control violation sequence and the test-based technique are started at the same time (Step 3). Then it is observed whether violation events are detected by the test-based measurement technique (Step 4). Provided the sample size is sufficiently large, i.e., enough measurement results have been produced (Step 4), the parameters of the unknown parent distribution are inferred, that is, we draw conclusions about the general accuracy of the test-based technique under evaluation (Step 5). These inferences are considered valid with regard to the test and control violation configuration parameters.
5.3 SECURITY CONTROL VIOLATION

In this section, it is described how to violate controls of a cloud service which a test-based measurement technique aims to detect, that is, which the technique’s evidence is expected to indicate. Thereby, the ground truth is established which allows to reason about correctness of evidence produced by specific test-based measurement technique.

5.3.1 CONTROL VIOLATION SEQUENCE

Recall that one of the key drivers for continuously testing cloud services is founded on the assumption that a cloud service’s property is non-stationary, that is, may change over time where these changes can lead to control violations. This means that the properties of cloud service may comply with a control at some time while at other times, they do not.

In order to mock such non-stationary behavior of cloud services’ properties, control violations have to continuously, i.e., repeatedly create control violation events ($cve$) over time. During a $cve$, a cloud service’s properties are manipulated so that the service does not comply with the control, i.e., not to satisfy the SLOs and SQOs associated with the control. Between two successive $cve$, the cloud service’s properties satisfy meet relevant SLOs and SQOs. This control violation sequence can be described as follows:

$V = \{cve_1, cve_2, ..., cve_i\}.$

As Figure 5-5 shows, each $cve$ starts at $cve^s$ and ends $cve^e$, thus having a duration of

$cveD = cve^e - cve^s.$
where the service does not comply with the control. Furthermore, the time between two successive control violation events $cve_{i-1}$ and $cve_i$ is

$$cveW = cve_{i-1}^e - cve_i^e.$$

**Figure 5-5 Sequence of control violation events cve**

**5.3.2 CONTROL VIOLATION DESIGN**

The design of a control violation is driven by the specific control for whose validation the test-based measurement technique under evaluation aims to provide measurement results. Therefore, the question at this point is: Which properties of a cloud service have to be altered to violate a particular control?

It is important to note at this point that it is *not* the aim here to design control violations which are *complete*, that is, which manipulates a cloud service in *any* possible way such that a particular control is not satisfied. While such a complete control violation design would be helpful to evaluate the completeness of the test-based measurement technique, designing such a complete control violation faces similar challenges to deriving suitable test metrics from high-level, ambiguous SLO and SQO definition: To that end, interpreting what it means for a specific control to be satisfied or dissatisfied on the implementation level of a cloud service instance is needed. The difference to deriving test metrics is, however, that we were to design mechanisms intentionally manipulating a cloud service’s properties to violate the control.

The goal of our evaluation is *correctness* of a test-based measurement technique, that is, the goal is to evaluate how accurate and precise the results produced by the test-based technique under evaluation are. Therefore, the test configuration of the continuous test under evaluation can serve as a starting point to derive the design of the control violation.

The control violation design process consists of two major steps:
1. **Inspect assert parameter:** The first step consists of inspecting the configuration of the test-based measurement technique under evaluation. Recall that a single test result of a test-based technique test is produced by executing a test suite which fails if any test case bound the test suite fail (see Deliverable 3.2, Chapter 4 for further detail). Therefore, the assert parameters which are used to configure the expected outcome of each test case are inspected. Based on the assert parameters and on their configured value, it can be determined which property of the cloud service has to be manipulated in order for these asserts to not be satisfied.

Consider, as an example, that a test-based measurement technique probes a set of ports to check if the cloud service exposes sensitive interfaces. The assert parameters of the test definition will denote the ports which are considered sensitive, that is, should not be reachable. A control violation event may, e.g., manipulate the service’s properties such that it exposes the blacklisted ports.

2. **Specify control violation events:** The second step consists of deciding on the lower \((cveW_L)\) and upper \((cveW_R)\) limit of the interval between two successive control violation events \(cveW\). Furthermore, the lower \((cveD_L)\) and upper \((cveD_R)\) limit of the time during which a cloud service’s property is manipulated to render it non-compliant have to be defined. The following section explains the purpose of randomizing duration of and interval between control violation events. Note that deciding on how many control violation events a control violation sequence should consist of is driven by the selected precision measures which are explained in detail in Section 5.4.

### 5.3.3 STANDARDIZING CONTROL VIOLATION EVENTS

Control violation sequences establish the ground truth against which specific test-based measurement techniques are evaluated. To infer conclusions about the general accuracy of a test-based measurement technique, ideally any possible sequence of any possible control violation event has to experimentally evaluated. Naturally, this is infeasible in practice and a sequence of control violation events \(V\) has to be selected which meets tolerable time and space constraints.

But how to select a sequence \(V\) which allows to draw conclusions about the general correctness of a test-based technique? The answer consists of two parts: At first, a control violation event needs to be standardized: For each \(cve\) we use to construct \(V\), the duration of the control violation \(cveD\) and the waiting time before start \(cveW\) are selected randomly from intervals \([cveD_L, cveD_R]\) and \([cveW_L, cveW_R]\), respectively. Choosing these intervals’ limits permits to configure control violations according to tolerable space and time limitations. Secondly, it
needs to be decided how many \( cve \), i.e., \(|V|\) are required to infer conclusions about the general accuracy and precision of the test-based measurement technique test under evaluation. This depends on the statistical inference method which, in turn, depends on the precision measure. This is addressed for each precision measure in the following Section.

5.4 ACCURACY AND PRECISION MEASURES

This section describes models to estimate the accuracy and precision of test-based measurement techniques. Hereafter, these models are referred to as accuracy measures and precision measures. These measures are based on the universal test metrics \( brC, fpsC, fpsD, \) and \( cfpsD \) introduced in Section 5.1.1.

In order to derive the accuracy and precision measures, each of the next four sections (5.4.1–5.4.4) follow these three steps:

1. **Evaluate measurement results**: The measurement results produced by a test-based measurement technique during a control violation sequence are used to evaluate to determine whether they are correct or erroneous. In the latter case, the type of observed error is specified which depends on the universal test metric used, e.g., a false negative basic test result incorrectly suggesting that a cloud services does not satisfy a control.

2. **Derive accuracy measures**: Using the evaluation of the measurement results as input, the accuracy measures then estimate if and how the measured values produced by test-based measurement techniques under evaluation deviate from the accepted, i.e., true values as established by control violation sequences.

3. **Derive precision measures**: Based on the evaluation measures, the precision measures estimate of and how the measured values spread about the accepted value.

5.4.1 BASIC-RESULT-COUNTER

This section describes how to estimate accuracy and precision of measurement results using the Basic-Result-Counter test metric \( (brC) \). To that end, the next section describes the evaluation of measurement results using different evaluation measures. Thereafter, it is detailed how to use these evaluation measures to compute accuracy and precision measures.
EVALUATION OF MEASUREMENT RESULTS

Hereafter, it is explained how to use the Basic-Result-Counter metric ($brC$) to evaluate a test-based measurement technique. To that end, we check whether measurement results correctly indicated absence or presence of a control violation event. Recall that $brC^F$ and $brC^T$ count failed $br^F$ and passed test results $br^T$, respectively. Furthermore, each test $tsr$ producing a basic test result $br$ starts at $tsr^s$ and ends at $tsr^e$, having a test duration of $tsrD$.

- **True negative basic test result counter ($brC^{TN}$):** A test produces a true negative result if the test fails at a time when a control is violated. As shown in Figure 5-6, a $brC^{TN}$ is produced if a failing test starts ($tsr^s$) after a control violation event starts ($cve^s$) and the test ends ($tsr^e$) before the event ends ($cve^e$):

$$brC^{TN} = cve^s \leq tsr^s \land tsr^e \leq cve^e.$$

We count any the true negative test results observed during the control violation sequence. As a result, we obtain $brC^{TN}$.

![Figure 5-6 True negative basic test result ($brC^{TN}$)](image)

- **True positive basic test result counter ($brC^{TP}$):** A true positive test result is produced if the test passes at a time when no control is violated. As shown in Figure 5-7, a passing test producing a true positive result starts after the previous control violation event ends and ends before the next control violation event starts:

$$brC^{TP} = cve^e < tsr^s \land tsr^e < cve^{s+1}_i.$$

There are two special cases: First, a test which passes prior to any control violation event is a true positive. Therefore, any passing test which ends ($tsr^e$) before the first violation event starts ($cve^e_i$) is a true positive:

$$brC^{TP} = tsr^e < cve^e_i.$$
Second, a test that passes after the last control violation even is a true positive test result. Thus any passing test which starts \((tsr^s)\) after the last control violation event \(j\) ends \((cve^e_j)\) is a true positive:

\[
br^{TP} = cve^e_j < tsr^s.
\]

Any true positive basic test result which is observed during a control violation sequence is counted using \(br^{C^{TP}}\).

**Figure 5-7 True positive basic test result \((br^{TP})\)**

- **False negative basic test result counter \((br^{C^{FN}})\):** If a test fails at a time when no control is violated, then the test produces a false negative test result. When comparing Figure 5-7 and Figure 5-8, it becomes evident that the definition of a false negative test result is analogous to the definition of a true positive test result. The only difference being that the test result incorrectly fails:

\[
br^{FN} = cve^e_i < tsr^s \land tsr^e < cve^s_{i+1}.
\]

Furthermore, similar to true positive results, two special cases exist: First, a test that incorrectly fails prior to any control violation event is a false negative. Therefore, any failing test which ends \((tsr^e)\) before the first violation event starts \((cve^s_i)\) is a false negative:

\[
br^{FN} = tsr^e < cve^s_i.
\]

Second, a test that incorrectly fails after the last control violation event is a false negative test result. Therefore, any failing test which starts \((tsr^s)\) after the last control violation event \(j\) ends \((cve^e_j)\) is a false negative:

\[
br^{FN} = cve^e_j < tsr^s.
\]

Any false negative basic test result which are observed during a control violation sequence is counted using \(br^{C^{FN}}\).
Figure 5-8 False negative basic test result \( (br_{FN}) \)

- **False positive basic test result counter \( (br_{FP}) \):** If a test passes at a time when a control is violated, then the incorrectly passing test produces a false positive result \( (br_{FP}) \). The definition of \( br_{FP} \) is similar to a true negative result (see Figure 5-6), only that the test incorrectly passes:

\[
br_{FP} = cve^s \leq tsr^s \land tsr^e \leq cve^e.
\]

Figure 5-9: False positive basic test result \( (br_{FP}) \)

Also, there is one special case: As shown in Figure 5-9, a passing test may cover one or more control violation events completely:

\[
br_{FP} = cve_i^e < tsr^s \land tsr^s < cve_{i+1}^s \land cve_{i+j}^e < tsr^e \land tsr^e < cve_{i+j+1}^e.
\]
We count all false positive results using $br_{C^{FP}}$.

- **Pseudo true negative basic test result counter ($br_{C^{PTN}}$):** Similar to a true negative test result, a test produces a pseudo true negative result if it fails at a time when a control is violated. However, unlike a $br_{TN}$, a $br_{PTN}$ is produced by a test only partially overlapping with the control violation event. There are two cases of partial overlapping to take into account:

  1. **Failing test ends during control violation event:** A $br_{PTN}$ is produced by a failing test which starts ($tsr^S$) prior to the start of the control violation event ($cve^S$). Furthermore, the test ends ($tsr^E$) after the violation events starts ($cve^E$) and before the control violation ends ($cve^e$):

     $$br_{PTN} = tsr^S < cve_i^S \land cve_i^S \leq tsr^E \land tsr^E \leq cve_i^e.$$

     Consider, as an example, the following scenario: A test starts measuring available bandwidth of a virtual machine. Only after the test started, the limitation of bandwidth of the virtual machine is induced by a control violation event. Thus, while at the beginning of the test no control was violated, later during the test it was. If the measurement result in total determines that the available bandwidth was insufficient, then the test fails, producing a pseudo true negative result $br_{PTN}$.

  2. **Failing test starts during control violation event:** A $br_{PTN}$ is produced by a failing test which starts ($tsr^S$) after a control violation event starts ($cve^E$) and starts before the control violation event ends ($cve^e$). Further, the test only ends ($tsr^e$) after the violation events ends ($cve^E$):

     $$br_{PTN} = cve_i^S \leq tsr^S \land tsr^S \leq cve_i^e < tsr^e.$$
Figure 5-11 shows a $br^{PTN}$ where a correctly failing test ends during a control violation event and Figure 5-12 depicts the case where a correctly failing test starts during a control violation event. In Figure 5-11, note the dotted line between the start of the test ($tsr^s$) and the start of the violation event ($cve^s$). It indicates that a test can cover multiple control violation events. Similarly, in Figure 5-12, the dotted line between the end of the control violation event ($cve^e$) and the end of the test ($tsr^e$) indicates that the test may cover multiple control violation events.

If a test covers multiple $cve$, then this implies that a test takes longer to complete ($tsrD$) than the duration of the control violation event ($cveD_i$), that is, $tsrD > cveD_i$.

**Figure 5-11 Pseudo true negative basic test result ($br^{PTN}$)**

**Figure 5-12 Pseudo true positive basic test result ($br^{PTN}$)**

Lastly, $br^{PTN}$ counts any occurrence of pseudo true negative test results.

- **Pseudo false positive basic test result counter ($br^{PFP}$):** A test produces a pseudo false positive result if the test partially overlaps with a control violation event but incorrectly passes. This means that the definition of $br^{PFP}$ is identical to $br^{PTN}$, the only difference being that the test result is positive. As in the case of a $br^{PTN}$, a
\( br^{PFP} \) can end during a control violation event or it can start during a control violation event. Also, a \( br^{PFP} \) may cover multiple control violation events. The number of occurrences of pseudo false positive results are counted using \( br^{C_{PFP}} \).

**ACCURACY MEASURES BASED ON BRC**

The previous paragraph introduced six evaluation measures based on the Basic-Result-Counter (\( brC \)) which serve to analyze the measurement results produced by a test-based measurement technique under evaluation during a control violation sequence. To summarize:

- True positive basic test result counter \( (br^{C_{TP}}) \),
- true negative basic test result counter \( (br^{C_{TN}}) \),
- false negative basic test result counter \( (br^{C_{FN}}) \),
- false positive basic test result counter \( (br^{C_{FP}}) \),
- pseudo true negative basic test result counter \( (br^{C_{PTN}}) \), and
- pseudo false positive basic test result counter \( (br^{C_{PFP}}) \).

These evaluation measures are used as input to compute accuracy measures. To that end, we draw on standard accuracy measures used in binary classification described by, e.g., (14), (15) and (16). Hereafter, it is described which specific measures are selected and how to interpret them to evaluate the accuracy of test-based measurement techniques.

- **Overall accuracy** \( (oac) \): The measure delineates the ratio between all correctly passed or failed tests \( (br^{C_{TN}} + br^{C_{PTN}} + br^{C_{TP}}) \) and all observed test results \( (br^{C_{TN}} + br^{C_{PTN}} + br^{C_{FN}} + br^{C_{TP}} + br^{C_{FP}} + br^{C_{PFP}}) \). The overall accuracy permits to evaluate out of all observed measurement results of a test-based technique under evaluation, how many are correct results:

\[
oac^{brC} = \frac{(br^{C_{TN}} + br^{C_{PTN}} + br^{C_{TP}})}{(br^{C_{TN}} + br^{C_{PTN}} + br^{C_{FN}} + br^{C_{TP}} + br^{C_{FP}} + br^{C_{PFP}})}
\]

- **True negative rate** \( (tnr) \): This measure delineates the proportion of correctly failed tests \( (br^{C_{TN}} + br^{C_{PTN}}) \) out of any test that should actually have failed \( (br^{C_{TN}} + br^{C_{PTN}} + br^{C_{FP}} + br^{C_{PFP}}) \). Using \( tnr \), the ability of a test-based technique to correctly detect if a cloud services complies with a control or not can be analyzed:

\[
tnr^{brC} = \frac{(br^{C_{TN}} + br^{C_{PTN}})}{(br^{C_{TN}} + br^{C_{PTN}} + br^{C_{FP}} + br^{C_{PFP}})}
\]
• **True positive rate (tpr):** This measure describes the ratio between correctly passed tests \((brC^{TP})\) and all tests that were expected to pass \((brC^{TP} + brC^{FN})\). It permits to evaluate how well a test-based technique correctly indicates that a cloud service satisfies the control the test aims to check:

\[
tpr_{brC} = \frac{brC^{TP}}{(brC^{TP} + brC^{FN})}
\]

• **False negative rate (fnr):** This measure describes the ratio between incorrectly failed tests \((brC^{FN})\) and all tests that were expected to pass \((brC^{TP} + brC^{FN})\). Based on this measure, we can evaluate how often a test-based technique incorrectly suggests that a control is not fulfilled by a cloud service:

\[
fnr_{brC} = \frac{brC^{FN}}{(brC^{TP} + brC^{FN})} = 1 - tpr_{brC}.
\]

• **False positive rate (fpr):** This measure describes the ratio between incorrectly passed tests \((brC^{FP} + brC^{PFP})\) and all observed tests that actually should have failed \((brC^{TN} + brC^{PTN} + brC^{FP} + brC^{PFP})\). It permits to describe the proportion of a test-based technique’s results which incorrectly suggest that a control of a cloud service is fulfilled:

\[
fpr_{brC} = \frac{(brC^{FP} + brC^{PFP})}{(brC^{TN} + brC^{PTN} + brC^{FP} + brC^{PFP})} = 1 - tnr_{brC}.
\]

• **False discovery rate (fdr):** This measure captures the ratio between incorrectly passed tests \((brC^{FP} + brC^{PFP})\) and all test which passed \((brC^{TP} + brC^{FP} + brC^{PFP})\). This allows us to reason about how often (out of all observed positive test results) measurement results of a test-based technique should have indicated failure, that is, measurement results which incorrectly indicated that a cloud service satisfies a control:

\[
fdr_{brC} = \frac{(brC^{FP} + brC^{PFP})}{(brC^{FP} + brC^{TP} + brC^{PFP})} = 1 - ppv_{brC}.
\]

• **Positive predictive value (ppv):** This measure delineates the ratio between correctly passed tests \((brC^{TP})\) and all test that passed \((brC^{TP} + brC^{FP} + brC^{PFP})\). Using this measure, it is possible to quantify the proportion of measurement results within all positive results which correctly suggest that a cloud service meets a control:

\[
ppv_{brC} = \frac{brC^{TP}}{(brC^{TP} + brC^{FP} + brC^{PFP})} = 1 - fdr_{brC}.
\]

• **False omission rate (for):** This measure describes the ratio between incorrectly failed tests \((brC^{FN})\) and all tests which failed \((brC^{TN} + brC^{PTN} + brC^{FN})\). This makes it is
possible to describe the proportion of measurement results produced by a test-based technique that should have passed within all produced test result that failed:

\[

\text{for}^{\text{brC}} = \frac{\text{brC}^{\text{FN}}}{(\text{brC}^{\text{TN}} + \text{brC}^{\text{PTN}} + \text{brC}^{\text{FN}})} = 1 - \text{npv}^{\text{brC}}.

\]

- **Negative predictive value (npv):** This measure describes the ratio between correctly failed tests \((\text{brC}^{\text{TN}} + \text{brC}^{\text{PTN}})\) and all tests that failed \((\text{brC}^{\text{TN}} + \text{brC}^{\text{PTN}} + \text{brC}^{\text{FN}})\). This allows to capture the proportion of results produced by a test-based technique which correctly indicate that a cloud service does not meet a control:

\[

\text{npv}^{\text{brC}} = \frac{(\text{brC}^{\text{TN}} + \text{brC}^{\text{PTN}})}{(\text{brC}^{\text{TN}} + \text{brC}^{\text{PTN}} + \text{brC}^{\text{FN}})} = 1 - \text{for}^{\text{brC}}.

\]

**PRECISION MEASURES BASED ON BRQ**

All accuracy measures based on evaluating basic test results \((\text{br})\), e.g., true negative rate \((\text{tnr})\), false positive rate \((\text{fpr})\), and negative predictive value \((\text{npv})\) have in common that they are proportions, that is, they provide the fraction of, e.g., correct test results of any observed test results. Thus we can construct confidence intervals for these proportions, that is, estimate the precision of these accuracy measures using interval estimates.

Consider, as an example, computing a confidence interval of 95\% for \(\text{npv}^{\text{brC}}\). This interval estimate allows statements such as \textit{we are 95\% confident that the npv}^{\text{brC}} \textit{of a test-based technique under evaluation is contained in the interval}. This inference is valid with respect to the configuration of the test-based technique and the control violation sequence.

Continuing our example for \(\text{npv}^{\text{brC}}\), we compute this interval estimate with

\[

\text{npv}^{\text{brC}} \pm z_{95\%} \times se_{\text{npv}}.

\]

\(z_{95\%}\) is the value that separates the middle 95\% of the area under the standard normal (or \(z\)) distribution, and \(se\) is the standard error which can be estimated with

\[

se_{\text{npv}} = \sqrt{\frac{\text{npv}^{\text{brC}} \times (1 - \text{npv}^{\text{brC}})}{n}}.

\]

\(\text{npv}^{\text{brC}}\) makes an educated guess of \(\text{npv}\) proportion in the parent distribution. If no historical information on \(\text{npv}^{\text{brC}}\) of the parent distribution is available, then \(\hat{\text{npv}}^{\text{brC}} = 0.5\) can be chosen denoting the conservative option. Further, \(n\) is the sample size which in this example for \(\text{npv}^{\text{brC}}\) consists of any basic failed test result used to compute \(\text{npv}^{\text{brC}}\), that is,

\[

n = \text{brC}^{\text{TN}} + \text{brC}^{\text{PTN}} + \text{brC}^{\text{FN}}.

\]
As stated above, the standard normal distribution is used to look up the value for $z_{95\%}$. This requires the sampling distribution of the proportion to be Gaussian. Determining the required sample size $n$, the margin of error $E_{npv}^{95\%} = z_{95\%} \times se$ is solved for the sample size $\bar{n}$:

$$\bar{n} = \frac{z_{95\%} \times \bar{npv}^{brC} \times (1 - \bar{npv}^{brC})}{E^2}$$

where $E$ delineates the desired margin of error.

Recall that in Section 5.3.2 and 5.3.3, the question was brought forward how many control violation events $|V|$ are needed to infer conclusions about the general accuracy of a test-based measurement technique under evaluation. Continuing the example for $npv^{brC}$, determining the required size of $V$ can be formulated as an optimization problem:

$$\begin{align*}
\text{minimize} & \quad |V| \\
\text{subject to} & \quad \bar{n} \leq br{C^{TN}} + br{C^{PTN}} + br{C^{FN}}
\end{align*}$$

Thus at least as many control violation events $cve$ have to be induced as are required to observe $\bar{n}$ test results. Following the above steps, interval estimates for the remaining accuracy measures, i.e., $oac^{brC}$, $tnr^{brC}$, $tpr^{brC}$, $mr^{brC}$, $pr^{brC}$, $dr^{brC}$, $ppv^{brC}$, and $orr^{brC}$ introduced in Section 5.4.1 can be computed analogously.

### 5.4.2 FAILED-PASSED-SEQUENCE-COUNTER

This section describes how to estimate the accuracy and precision of a test-based measurement technique under evaluation using the Failed-Pass-Sequence-Counter metric ($fpsC$). To that end, the next section describes the evaluation of test results using three evaluation measures. Thereafter, it is described how we leverage these evaluation measures to compute accuracy and precision measures.

**EVALUATION OF MEASUREMENT RESULTS**

This section explains how to evaluate a test-based technique based on the Failed-Passed-Sequence-Counter metric ($fpsC$). Recall that $fpsC$ counts the occurrence of failed-passed-sequences ($fps$), it is a special sequence of basic test results which starts with a failed test and ends with the next passing test (see Section 5.1.1 for further detail). A $fps$ aims at detecting temporal control violations, that is, control violations that persist for some time. In order to
evaluate the measurement results of a test-based technique, we inspect if and how any fps overlaps with control violation events cve.

- **True negative fps** ($fps^{TN}$): A fps that consists of only correct basic test results, i.e., true negative test results ($br^{TN}$), pseudo true negative test results ($br^{PTN}$) and one final true positive test result ($br^{TP}$). A $fps^{TN}$ starts ($fps^{s}$) after the last control violation event ends ($cve^{e}_{i-1}$) and starts before the next control violation event ends ($cve^{e}_{i}$). Furthermore, the $fps^{TN}$ ends ($fps^{e}$) only after the next control violation ends ($cve^{e}_{i+1}$). Formally, we can define a true negative fps as follows:

$$fps^{TN} = cve^{e}_{i-1} \leq fps^{s} \land fps^{s} \leq cve^{e}_{i} \land cve^{e}_{i} < fps^{e}.$$ 

Note that a $fps^{TN}$ may cover multiple cve. Figure 5-13 shows an exemplary true negative fps whose first failed test produced a pseudo true negative result ($br^{PTN}$) which starts at $t_{srj}$. This example $fps^{TN}$ covers two control violation events, that is, $cve_{i}$ and $cve_{i+1}$. $fps^{CTN}$ counts the number of $fps^{TN}$ observed during a control violation sequence.

**Figure 5-13 True negative failed-passed-sequence ($fps^{TN}$)**

Note that a true negative fps which detects the first control violation event during experimental evaluation depicts a special case: If no previous cve exists, then the following, simplified definition of $fps^{TN}$ applies:

$$fps^{TN} = fps^{s} \leq cve^{e}_{i} \land cve^{e}_{i} < fps^{e}.$$ 

- **False negative fps** ($fps^{FN}$): A fps that consists of at least one incorrect basic test result, i.e., false negative test results ($br^{FN}$) or false positive test result ($br^{FP}$) or both. A basic variant of an $fps^{FN}$ is observed if any failed basic test results are false negatives and only the last test passes correctly. In this case, the fps starts after the last cve ends ($cve^{e}_{i}$) and ends ($fps^{e}$) before the next cve starts ($cve^{e}_{i+1}$):

$$fps^{FN} = cve^{e}_{i} < fps^{s} \land fps^{e} < cve^{e}_{i+1}.$$
Figure 5-14 shows this basic version of a $fps^{FN}$. We define $fps^{C_{FN}}$ which counts any occurrence of $fps^{FN}$ observed during a control violation sequence.

![Figure 5-14 False negative fps](image)

**Figure 5-14 False negative fps**

However, false negative $fps$ may also contain true negative basic test results. This is the case if after a $cve$ ended and before the next $cve$ starts, that is, no control violation event is induced, basic results still incorrectly indicate a control violation. Figure 5-15 shows an example case of this error: After the control violation event $cve_i$ ended at $cve_i^e$ and before the next $cve$ starts at $cve_{i+1}^s$, the test $tst_{j+1}$ produces a false negative test result at $tst_{j+1}^e$.

![Figure 5-15 False negative failed-passed-sequence (fps^{FN}) with true negative and false negative basic test result (br^{TN} & br^{FN})](image)

**Figure 5-15 False negative failed-passed-sequence ($fps^{FN}$) with true negative and false negative basic test result ($br^{TN}$ & $br^{FN}$)**

Complementary indicators for $fps^{FN}$ are the false omission rate ($for^{brC}$) and negative predictive value ($npv^{brC}$). These accuracy measures are calculated using on basic test results (see Section 5.4.1). The more incorrect negative basic test results are observed during evaluation of a test-based technique, the higher $for^{brC}$ and the lower $npv^{brC}$.

At last, the last test of an $fps^{FN}$ can be a false positive, i.e., the last test result incorrectly indicates that the cloud services satisfies a control. Figure 5-16 shows one example of this error: After a test correctly failed at $tst_{j+1}^e$, the next test incorrectly passes while the control is still violated, thereby producing a false positive test result ($br^{FP}$) at $tst_{j+2}^e$. 

![Figure 5-16 False positive test result](image)
Figure 5-16 False negative failed-passed-sequence ($fps^{FN}$) with false positive basic test result ($br^{FP}$)

As a complementary means to investigate this type of error, we can use of the positive predictive value ($ppv^{brC}$) and false discovery rate ($fdr^{brC}$) introduced in Section 5.4.1: The more incorrect positive basic test results are observed during evaluation, the higher $fdr^{brC}$ and the lower $ppv^{brC}$.

- **False positive $fps$ ($fps^{FP}$):** A $fps$ indicates that a cloud service does not satisfy a control over time. Thus, a control violation event not detected by a test-based measurement technique is considered false positive $fps$. Figure 5-17 shows a $cve$ that starts after the last $fps$ ended ($fps^s_j$) and ends before the next $fps$ starts ($fps^s_{j+1}$):

$$fps^{FP} = fps^s_j < cve^e \land cve^e < fps^s_{j+1}.$$  

We use $fps^{FP}$ to count the occurrences of $fps^{FP}$ during a control violation sequence.

Figure 5-17 False positive $fps$

**ACCURACY MEASURES BASED ON FPSC**

The previous paragraphs introduced three evaluation measures derived from the Failed-Passed-Sequence-Counter ($fpsC$):

- True negative Failed-Passed-Sequence-Counter ($fps^{TN}$),
- false negative Failed-Passed-Sequence-Counter ($fps^{FN}$) and
- false positive Failed-Passed-Sequence-Counter ($fps^{FP}$).
These evaluation results are now used to calculate accuracy measures. To that end, analogous to the accuracy measures based on brC introduced in Section 5.4.1, standard measures used in binary classification are leveraged. The following paragraphs explain which measures are selected and how these measures can be used to interpret the accuracy of a test-based measurement technique under evaluation to identify temporal violations of controls.

- **True negative rate (tnr):** This measure describes the ratio between correctly detected control violation events ($fpsC^{TN}$) and all control violation events that were induced by the control violation sequence, that is, which could have been detected ($fpsC^{TN} + fpsC^{FP}$):

  \[
  tnr^{fpsC} = \frac{fpsC^{TN}}{(fpsC^{TN} + fpsC^{FP})} = 1 - fpr^{fpsC}.
  \]

  $tnr^{fpsC}$ allows to evaluate how well a test-based measurement technique works in detecting intervals when a control is not satisfied by a cloud service.

- **False positive rate (fpr):** This measure describes how many control violation events were not detected ($fpsC^{FP}$) out of all events that could have potentially been detected ($fpsC^{TN} + fpsC^{FP}$):

  \[
  fpr^{fpsC} = \frac{fpsC^{FP}}{(fpsC^{TN} + fpsC^{FP})} = 1 - tnr^{fpsC}.
  \]

  Based on $fpr^{fpsC}$, the proportion can be described how many control violation events were missed by test-based technique under evaluation. It is the percentage of how many times the test-based technique failed to indicate that a control is not satisfied by a cloud service.

- **False omission rate (for):** This measure captures the ratio of incorrectly detected control violation events ($fpsC^{FN}$) and all control violation events that a test-based technique indicated ($fpsC^{TN} + fpsC^{FN}$):

  \[
  for^{fpsC} = \frac{fpsC^{FN}}{(fpsC^{TN} + fpsC^{FN})} = 1 - npv^{fpsC}.
  \]

  Using $for^{fpsC}$, it is possible to make statements about how often a test-based technique incorrectly suggested that a cloud service did not comply with a control for some time out of all detected control violation events.
• **Negative predictive value (npv):** This measure delineates the ratio between any correctly detected control violation event \(fps CTN\) and all detected control violation events \(fps CTN + fps CFN\):

\[
npv_{fps} = \frac{fps CTN}{fps CTN + fps CFN} = 1 - for_{fps}.
\]

On the basis of \(npv_{fps}\), it can be evaluated how many times a test-based measurement technique correctly indicated a control violation event out of all control violation events that the test-based technique suggested.

**PRECISION MEASURES BASED ON (FPSC)**

Analogous to the accuracy measures derived from basic test results, the accuracy measures \(tnr_{fps}, pr_{fps}, or_{fps}\) and \(npv_{fps}\) can be treated as proportions. Therefore, we apply the same idea proposed in the previous section to calculate interval estimates for \(tnr_{fps}, pr_{fps}, or_{fps}\) and \(npv_{fps}\) in order to infer general statements about the accuracy of a test-based measurement technique based on \(fpsC\).

Note that there exists one important difference to the approach described in the previous section: At least as many control violation events \(|V|\) have to be induced as are needed to observe \(\tilde{n} fps\) during the control violation. Consider, as an example, that we want to construct a confidence interval for \(tnr_{fps}\). The sample size \(n\) for \(tnr_{fps}\) consists of any control violation event which should have been detected by the test-based measurement technique, that is,

\[n = fps CTN + fps CFP.\]

The corresponding optimization problem to find the required sample size \(\tilde{n}\) for \(tnr_{fps}\) thus can be formulated as follows:

\[
\text{minimize } |V|\\
\text{subject to } \tilde{n} \leq fps CTN + fps CFP
\]

Precision estimates for the remaining three accuracy measures, i.e., \(pr_{fps}, or_{fps}\) and \(npv_{fps}\) can be computed analogously by following the above steps.
This section describes how to estimate the accuracy and precision of a test-based technique based on measurement results produced by the universal test metric *Failed-Passed-Sequence-Duration* (*fpsD*). First, the different types of errors that an *fpsD* may make when attempting to determine the duration, start and end of a control violation event are described. Thereafter, it is explained how to use these evaluation measures to estimate the accuracy and precision of a test-based measurement technique based on *fpsD*.

**EVALUATION OF MEASUREMENT RESULTS**

The following paragraphs describe how to evaluate a test-based measurement technique based on measurement results produced by the *Failed-Passed-Sequence-Duration* test metric (*fpsD*). Recall that *fpsD* captures the time (e.g., in milliseconds) between the start of the first failed test (*fps*), i.e., first element of a fps, and the start of the next subsequent passed test (*fps*), i.e., last element of a fps (see Section 5.1.1).

- **Duration error of true negative *fpsD* (*efpsD*):** Having observed a true negative *fps*, the difference between the duration of the *fps*, i.e., *fpsD = fps* - *fps* and the duration *cveD* of any control violation events which is covered by the *fps* is calculated. Figure 5-18 shows that a *fps* may cover multiple *cve*, however, it can, at most, cover all *cve* contained in the sequence *V* of the control violation sequence:

  \[
  fpsD_{TN} = fpsD_{TN} - \sum_{i=1}^{\vert V \vert} cveD_i .
  \]

  Note that we do not calculate the absolute difference between *fpsD* and covered *cveD*. This permits us to determine whether a *fpsD* overestimates or underestimates the duration of a control violation event: In case of *efpsD* > 0, then the *fpsD* overestimates the duration of covered control violation events (Figure 5-18). Otherwise, if *efpsD* < 0, then *fpsD* underestimates the duration of the control violation event (Figure 5-19). Lastly, if *efpsD* = 0, the *fpsD* and the duration of the covered control violation events are equal.
**Figure 5-18 True negative Failed-Passed-Sequence-Duration (fps\textsuperscript{TN}) which overestimates total duration of \textit{cve}_i and \textit{cve}_{i+1}**

Furthermore, the relative error that a \textit{fps} makes when estimating the duration of covered control violation events is calculated as follows:

$$efpsD_{rel}^TN = \frac{|efpsD_{rel}^TN|}{\sum_{i=1}^{|V|} cveD_i}$$

**Figure 5-19 True negative Failed-Passed-Sequence-Duration (fps\textsuperscript{TN}) which underestimates duration of \textit{cve}_i**

- **Pre-duration error of true negative fpsD (fpsD\textsuperscript{TN}_{pre}):** Until now, we focused our error definition on the estimated duration of control violation events provided by a true negative fpsD. However, as Figure 5-20 illustrates, the start of a fpsD\textsuperscript{TN} which estimates the start of the control violation event can be inaccurate, i.e., \textit{cve} < fps. In order to capture this error, we compute the difference between the start of a fps, i.e., the start of the first failed test which detected a control violation event (fps\textsuperscript{s}), and the start of the control violation event (cve\textsuperscript{s}):

$$efpsD_{pre}^TN = fps^s - cve^s.$$ 

In case of fpsD\textsuperscript{TN}_{pre} > 0, then the fps starts only after the cve starts. Note that this case implies that the first failed test of the fpsD\textsuperscript{TN} produced a true negative test result (br\textsuperscript{TN}). Further, if fpsD\textsuperscript{TN}_{pre} < 0, then the fps starts before the cve starts. This case, in turn, implies that the first test produced a pseudo true negative test result (br\textsuperscript{PTN}).
Figure 5-20 True negative Failed-Passed-Sequence-Duration (fpsTN) with efpsD_{pre}^{TN} > 0 and efpsD_{post}^{TN} > 0.

- **Post-duration error on true negative fpsD (efpsD_{post}^{TN}):** Recall that the last basic test result of a true negative fps is always a true positive basic test result. This means that a fpsTN by definition only ends after the control violation event ends. Figure 5-20 shows efpsD_{post}^{TN} which is the resulting error the last test result of a fpsTN makes when determining the end of a control violation event. Describing this error, the difference between the end of a control violation event (cve\(e\)) and the end of the fps, i.e., the start of the last test which passed is computed:

\[
efpsD_{post}^{TN} = fps^e - cve^e.
\]

- **Duration error of false negative fpsD (efpsD_{FN}):** If a false negative fps is observed, then the entire duration of that fps is considered to be erroneous because it incorrectly indicates a duration of a control violation event. Figure 5-21 shows a fpsDFN which is defined as follows:

\[
efpsD_{FN} = fps^e - fps^s.
\]

Figure 5-21 False negative Failed-Passed-Sequence-Duration (fpsDFN)

- **Duration error of false positive fpsD (efpsD_{FP}):** If a control violation event is not detected by a fps at all, then this missed cve is considered a false positive fps.
Consequently, the duration of a false positive $fps$ equals the duration of the missed control violation event (Figure 5-22):

$$efpsD^{FP} = cve^e - cve^s.$$

$Figure 5-22$ caption False positive Failed-Passed-Sequence-Duration ($fpsD^{FP}$)

**ACCURACY MEASURES BASED ON EFPFD**

In the previous section, we introduced five error types derived from the Failed-Passed-Sequence-Duration ($fpsD$) test metric:

- Duration error of true negative Failed-Passed-Sequence-Duration ($efpsD^{TN}$),
- pre-duration error of true negative Failed-Passed-Sequence-Duration ($efpsD^{TN}_{pre}$),
- post-duration error of true negative Failed-Passed-Sequence-Duration ($efpsD^{TN}_{post}$),
- duration error of false negative Failed-Passed-Sequence-Duration ($efpsD^{FN}$), and
- duration error of false positive Failed-Passed-Sequence-Duration ($efpsD^{FP}$).

When evaluating a test-based measurement technique, then instances of any of the above errors may be observed. These observations for each type of error on $fpsD$ can be treated as separate distributions: After having executed a control violation sequence and the test-based measurement technique under evaluation, it can be expected to obtain at most five distributions. However, in practice, a test-based technique may not produce any incorrect test results, i.e., neither $b^F_{FN}$ nor $b^F_{FP}$. This means that neither instances of $efpsD^{FN}$ not instances of $efpsD^{FP}$ are observed. However, a test-based technique which does not make any error on estimating the total duration, the start and the end of any control violation event is rather unlikely. The reason for this is that not observing any instance of $efpsD^{TN}$, $efpsD^{TN}_{pre}$, or $efpsD^{TN}_{post}$ requires the test-based technique to always perfectly estimate duration, start and end of any control violation event. Thus, it is reasonable to expect to observe at least three distributions after having evaluated a test-based measurement technique, i.e., $efpsD^{TN}$, $efpsD^{TN}_{pre}$ and $efpsD^{TN}_{post}$.

In order to estimate the accuracy of a test-based technique when measuring temporal control violations (e.g., in milliseconds), the arithmetic mean ($\bar{x}$) for each of the observed distributions
is computed. For example, to compute the arithmetic mean for $fpsDTN$, we add any instances $i$ of $fpsDTN$ contained in the sequence $EFPSDTN$ and divide by the number of elements in $EFPSDTN$:

$$\bar{x}^{TN} = \frac{(fpsD_{1}^{TN} + fpsD_{2}^{TN} + \cdots + fpsD_{i}^{TN})}{|EFPSDTN|}.$$  

Using $\bar{x}^{TN}$, we can describe the average error a $fpsDTN$ makes when estimating the duration of a control violation event. Calculation and interpretation of the remaining four error types is analogous.

As a complementary measure, also the median ($\hat{x}$) is computed which is the middle value of an ordered list. The median is helpful when values of, e.g., $EFPSDTN_{pre}$ do not increase arithmetically, i.e., if the difference between consecutive values of an ordered list is not constant. Consider, as an example, having observed $EFPSDTN_{pre} = (-8, -5, 10)$. The mean is $\bar{x}_{pre}^{TN} = -1$ while median tells us $\hat{x}_{pre}^{TN} = -5$.

**PRECISION MEASURES BASED ON EFPSD**

Describing the precision of a test-based measurement technique under evaluation, the following statistics are computed:

- **Standard deviation** ($sd$): This statistic measures the dispersion of values within a distribution. Drawing on the example from the previous paragraph, the standard deviation of the values in $EFPSDTN$ describe how far values spread around its mean:

  $$sd_{TN} = \sqrt{\frac{1}{|EFPSDTN|} \times ((fpsD_{1}^{TN} - \bar{x}_{TN})^2 + \cdots + (fpsD_{i}^{TN} - \bar{x}_{TN})^2)}$$

Using $sd$, it is possible to describe the variation of the different types of error which a test-based technique makes when measuring the duration of control violation events. Furthermore, the $sd$ can also be used to calculate the standard error of the mean which is needed to calculate confidence intervals which is explained in the following paragraph.

- **Confidence Interval for the sample mean**: In total, five types of errors were presented which a $fpsD$ may make when measuring the duration of a control violation event, e.g., $fpsD^{TN}$ and $fpsD^{FP}$. For each of these error types, the mean $\bar{x}$ of the observed distribution is computed serving as an accuracy measure. In order to make a general
statement about the precision of a test-based measurement technique, we can construct a confidence interval for each mean.

As an example, consider $efpsD^{TN}$, which captures the mean error that a test-based technique makes when determining the duration of a control violation event: A confidence interval on this mean permits us statements such as *we are 99% confident that the average error of a test-based measurement technique – with respect to the technique’s and control violation configuration – makes when estimating the duration of a control violation event is contained in the interval*. This estimate can be obtained as follows:

$$
\bar{x}_{TN} \pm t_{99\%} \times se_{\bar{x}}.
$$

$t_{99\%}$ is the value that separates the middle 99% of the area under the $t$-Distribution and $se$ is the standard error. $se$ can be estimated as follows:

$$
se_{\bar{x}} = \frac{sd_{TN}}{\sqrt{n}}.
$$

In context of the above example, the sample size $n$ is the number of observed true negative $fps$ and $sd$ is the standard deviation. In order to determine the required sample size $\tilde{n}$, the desired margin of error $\tilde{E}$ is solved for the sample size $\tilde{n}$, that is,

$$
\tilde{n} = \frac{\sigma^2 \times t_{99\%}^2}{E^2}.
$$

$\sigma^2$ is an educated guess of the population variance based on initial samples of $efpsD^{TN}$ or historical values.

Inferring statements about the general accuracy of a test-based measurement techniques based on the mean of, e.g., $\bar{x}_{TN}$ requires inducing a minimum number of control violation events. In our example for $efpsD^{TN}$, the minimum size of $V$ can be obtained by solving the following optimization problem:

**minimize** $|V|

**subject to** $\tilde{n} \leq fpsD^{TN}$

This means that at least as many control violation events need to be induced which are needed to observe $\tilde{n}$ $fpsD^{TN}$. Using these above steps, interval estimates for the means of $efpsD^{TN}_{pre}$, $efpsD^{TN}_{post}$, $efpsD^{FN}$, and $efpsD^{FP}$ can be calculated analogously.
Also, the minimum and maximum \((\text{min} \ \& \ \text{max})\) are computed, that is, the smallest and largest value for any type of error that was observed during evaluation of a test-based measurement technique. Using these statistics, most extreme errors that a test-based technique makes when measuring duration of control violation events can be described. Furthermore, comparing \text{min} and \text{max} to the standard deviation can help identifying if the measurement results produced by the test-based technique during evaluation contain outliers.

### 5.4.4 CUMULATIVE-FAILED-PASSED-SEQUENCE-DURATION

In this section, we describe how to determine the accuracy of a test-based measurement technique based on the universal test metric \textit{Failed-Passed-Sequence-Cumulative-Duration} \((\text{cfpsD})\). Hereafter, first the three evaluation measures \(\text{cfpsD}^\text{TN}\), \(\text{cfpsD}^\text{FN}\), and \(\text{cfpsD}^\text{FP}\) are introduced which are derived from \(\text{fpsD}^\text{TN}\), \(\text{fpsD}^\text{FN}\), and \(\text{fpsD}^\text{FP}\) observed during evaluation, respectively. Thereafter, it is explained how these evaluation measures can be leveraged to estimate the accuracy of a test-based technique under evaluation.

#### EVALUATION OF MEASUREMENT RESULTS

This section describes how to evaluate a test-based measurement technique based on the \textit{Failed-Passed-Sequence-Cumulative-Duration test metric} \((\text{cfpsD})\). Recall that this metric accumulates the value of any \(\text{fpsD}\) (e.g., in milliseconds) observed within a specified period of time. This allows to determine if a cloud service satisfies a control with temporal constraints within that period (see Section 5.1.1).

- **True negative \(\text{cfpsD} (\text{cfpsD}^\text{TN})\):** Each value of a true negative \(\text{fpsD}\) observed during evaluation of the test-based measurement technique is added, i.e.,

  \[
  \text{cfpsD}^\text{TN} = \text{fpsD}_1^\text{TN} + \text{fpsD}_2^\text{TN} + \cdots + \text{fpsD}_i^\text{TN}.
  \]

  This measure returns the total measured duration of correctly detected control violation events.

- **False negative \(\text{cfpsD} (\text{cfpsD}^\text{FN})\):** This evaluation measure holds the sum of any false negative \(\text{fpsD}\) which was produced by the test-based measurement technique under evaluation:

  \[
  \text{cfpsD}^\text{FN} = \text{fpsD}_1^\text{FN} + \text{fpsD}_2^\text{FN} + \cdots + \text{fpsD}_i^\text{FN}.
  \]

  \(\text{cfpsD}^\text{FN}\) captures the total measured duration of control violation events which the test-based technique incorrectly indicated.
• **False positive cpsD** \((cpsD^{FP})\): The sum of any false positive \(fpsD\) which was produced by the test-based technique under evaluation is computed by this metric:

\[
cpsD^{FP} = fpsD_1^{FP} + fpsD_2^{FP} + \cdots + fpsD_i^{FP}.
\]

Using \(cpsD^{FP}\), the total duration of control violation events that were *not* detected by the test-based measurement technique under evaluation can be described.

**ACCURACY MEASURES BASED ON (CFPSD)**

The previous three paragraphs introduced the following three evaluation measures:

- True negative Cumulative-Failed-Passed-Sequence-Duration \((fpsD^{T_{TN}})\)
- False negative Cumulative-Failed-Passed-Sequence-Duration \((fpsD^{F_{FN}})\)
- False positive Cumulative-Failed-Passed-Sequence-Duration \((fpsD^{F_{PP}})\)

In order to determine the overall accuracy of a test-based technique within a predefined period of time, that is, within the control violation sequence, the following three accuracy measures can be used:

- **Duration error of true negative cpsD** \((ecfpsD^{T_{TN}})\): This measure computes the difference between the cumulative duration of true negative \(fpsD\) and the total duration of any control violation event \(cve \in V\):

\[
ecfpsD^{T_{TN}} = fpsD^{T_{TN}} - \sum_{i=0}^{\mid V \mid} cveD_i.
\]

The accuracy measure \(ecfpsD^{T_{TN}}\) permits to describe if a test-based technique overestimates or underestimates the accumulated duration of control violations within a specified period of time. If the test-based technique overestimates the total duration of violated controls, then \(ecfpsD^{T_{TN}} > 0\). Otherwise, if the test-based technique underestimates the total duration of violated controls, then \(ecfpsD^{T_{TN}} < 0\). Lastly, if \(ecfpsD^{T_{TN}} = 0\), then the duration measured by the test-based technique perfectly matches the total duration of control violation events.

Furthermore, the ratio between \(ecfpsD^{T_{TN}}\) and the total duration of control violation events is computed:

\[
ecfpsD_{rel}^{T_{TN}} = \frac{|ecfpsD^{T_{TN}}|}{\sum_{i=1}^{\mid V \mid} cveD_i}.
\]
Using $ecfpsD^T_{FN}$, it is possible to describe the relative measurement error that a test-based technique makes when determining the total time during which a cloud service does not comply with a control.

- **Duration error of false negative $fpsD$ ($ecfpsD^{FN}$):** The total duration of false negative $fpsD$ that a test-based technique suggested is identical to the duration error of false negative $fpsD$, that is, $fpsD^{FN} = ecpfSD^{FN}$. However, the absolute total duration of a test-based technique’s measurement results incorrectly indicating temporary control violation provides only limited information because it lacks context. Therefore, we also compute the ratio between $ecfpsD^{FN}$ and the total amount of time during which the test-based technique indicated that the cloud service does not satisfy a control ($ecfpsD^{FN} + ecpfSD^{TN}$):

$$ecfpsD^{FN}_{rel} = \frac{fpsD^{FN}}{(fpsD^{FN} + fpsD^{FN})}$$

Based on $ecfpsD^{FN}_{rel}$, we can make statements about the proportion of detected temporary control violation which – out of the total duration of control violation events – was incorrect.

- **Duration error of false positive $fpsD$ ($ecfpsD^{FP}$):** The total duration of false positive $fpsD$ is identical to the duration error of false positive $fpsD$, i.e., $fpsD^{FP} = ecpfSD^{FP}$. Yet $ecfpsD^{FP}$ as an absolute value only provides the total amount of time where the test-based technique was expected to detect temporary control violation events but, in fact, it did not. In order to be able to assess the meaning of $ecfpsD^{FP}$, we relate it to total duration of control violation events as follows:

$$ecfpsD^{FP}_{rel} = \frac{fpsD^{FP}}{\sum_{i=1}^{V} cveD_i}$$

where $ecfpsD^{FP}_{rel}$ describes the proportion of control violation events’ duration which remained undetected in total.

**PRECISION MEASURES BASED ON CFPSD**

Recall the definition of precision presented in Section 5.1.2: Precision refers to closeness of agreement between successively measured values which implies that precision measures need at least two measured values as input. Since $ecfpsD^{TN}$, $ecfpsD^{FN}$, and $ecfpsD^{FP}$ are exactly calculated once after experimental evaluation of a test-based measurement technique, the concept of precision is not applicable drawing on $fpsD$. 
5.5 IMPLEMENTATION AND EXAMPLE EVALUATION

This section presents an example scenario in which we apply our method to evaluate and compare measurement results produced by a test-based measurement technique. The next section describes the components of our experimental setup. Thereafter, we present a scenario in which cloud service providers seek to evaluate tests to support continuous certification of cloud services according to controls related to the properties availability and security.

5.5.1 SETUP AND ENVIRONMENT

This section outlines the experimental setup used to evaluate measurement results produced by the test-based technique. We begin with the cloud service which is subject to testing. Then we briefly describe the control violation framework which is used to manipulate properties of the cloud services under test so that it does not comply with one or more controls as well as the test-based measurement technique. Finally, we present the evaluation engine which is used to computes the accuracy and precision measures presented in Section 5.4.

CLOUD SERVICES UNDER TEST

The cloud service under test consists of an instance of IaaS provided by OpenStack Mitaka\(^\text{17}\) on top of which an Apache\(^\text{18}\) web server is running. The virtual machine is equipped with 2 VCPUs and 4 GB RAM and running Ubuntu 16.04 server.

CONTROL VIOLATION FRAMEWORK

In order to trigger control violation events, a lightweight framework has been developed in Java which permits to repeatedly manipulate properties of a cloud service under test over time so that the service does not satisfy one or more controls for some time (for further detail see Section 5.3). The framework is extensible allowing to add novel control violation and multiple control violation sequences can be executed concurrently.

Each control violation event is persisted, including start and end time of each event, event duration as well as current iteration. This data serves as the reference which is later used by the evaluation engine (see paragraph below) to evaluate the accuracy and precision of measurement results produced by a test-based measurement technique. The control violation

\(^{17}\) https://www.openstack.org/software/mitaka/

\(^{18}\) https://httpd.apache.org/
framework is deployed on a designated virtual machine, attached to the identical tenant network as the cloud service under test.

**CONTINUOUS TEST-BASED MEASUREMENT TECHNIQUE**

The test is implemented following the framework introduced in Section 4 of Deliverable 3.2. The test is deployed on an external host, attached to a different network than the cloud services under test.

**EVALUATION ENGINE**

This component calculates accuracy and precision measures described in Section 5.4 as well as test and control violation statistics. For that purpose, the *Apache Commons Math* library is used. The evaluation engine is implemented in Java and runs locally on a personal computer and uses the control violation sequence’s data and produced test results as input.

**5.5.2 CONTINUOUSLY TESTING SECURE COMMUNICATION CONFIGURATION**

In this scenario, we consider a cloud service provider who, at the same time, acts a cloud service customer. This means that the provider offers a SaaS application to customers for whose delivery he leverages another cloud provider offering platform services (PaaS). Thus, components such as web server, data bases, and load balancer are supplied and maintained by the PaaS provider. Therefore, the SaaS provider cannot directly access the underlying applications and components but only has access to the necessary APIs. As a result, the PaaS provider is responsible to provide secure communication configurations which includes secure configuration of Transport Layer Security (TLS) used by the web server component of the SaaS application to deliver websites via HTTPS.

We assume that the SaaS provider seeks certification of his application according to controls which relate to property *secure communication configuration*. Examples for such cloud-specific controls are *KRY-02 Encryption of data for transmission (transport encryption)* of the Cloud Computing Compliance Controls Catalogue (BSI CS) (7), *EKM-03: Encryption & Key Management Sensitive Data Protection* of CSA’s Cloud Control Matrix (CCM) (1), and *A.14.1.2 Securing application services on public networks* of ISO/IEC 27001:2013 (17).

In order to support certification of his SaaS application, the provider want to utilize a continuous test-based measurement technique and configure it in such a way that it indicates
as accurately as possible if the secure communication configuration property of his SaaS application does not hold. This implies that the test-based technique should ideally detect any violation of the secure communication configuration property and the number of false positive measurement results produced by the technique should be as low as possible. Furthermore, if an insecure communication configuration is detected, then the SaaS provider seeks a test configuration which as accurately as possible detects how long the PaaS provider needs to remedy vulnerable communication configurations.

**ALTERNATIVE TEST CONFIGURATIONS**

In order to analyze TLS configurations of our cloud service under test, we leverage the tool sslyze\(^\text{19}\). Inspecting the output of sslyze permits to, e.g., find out whether the web server offers to communicate via known vulnerable cipher suites. If the web server does offer support for vulnerable cipher suites, then the secure communications configuration property of the cloud service under test does not hold which, in turn, leads to a violation of certificates' controls relating to this property.

The SaaS provider within our scenario can select one of the following three different candidate configurations for the test **TLSTest**:

- **TLSTest\(^{[0,10]}\)**: Each execution of **TLSTest** is triggered randomly in the interval \([0,10]\) after the last test completed.
- **TLSTest\(^{[0,30]}\)**: Each execution of **TLSTest** is triggered randomly in the interval \([0,30]\) after the last test completed.
- **TLSTest\(^{[0,60]}\)**: Each execution of **TLSTest** is triggered randomly in the interval \([0,60]\) after the last test completed.

No additional offset between test executions is configured while the number of successive iterations for all three **TLSTest** variants is set to infinity. Further, only measurement results produced during the control violation sequence are considered for evaluation.

**CONTROL VIOLATION CONFIGURATION**

For each TLSTest variant, we triggered 1000 vulnerable TLS configurations of the cloud service under test to evaluate the three candidate configurations of **TLSTest**. These vulnerable TLS configurations consist of altering the web server configuration such that it supports TLS communication using the weak cipher suite **TLS_RSA_WITH_DES_CBC_SHA**. Each event of an

\(^{19}\) https://github.com/nabla-c0d3/sslyze
insecure TLS configuration lasted at least 60 seconds plus selecting \([0,30]\) seconds at random. The interval between consecutive vulnerable configuration events lasted at least 120 seconds plus selecting \([0,60]\) seconds at random. Table 5-1 summarizes the control violation sequence statistics observed during experimental evaluation of \(TLSTest^{[0,10]}\), \(TLSTest^{[0,30]}\) and \(TLSTest^{[0,60]}\).

**Table 5-1 Summary of control violation sequence statistics for TLSTest**

<table>
<thead>
<tr>
<th>Sequence statistic (sec)</th>
<th>(\sqrt{\text{TLSTest}})</th>
<th>(0,10)</th>
<th>(0,30)</th>
<th>(0,60)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ccveD</td>
<td>75050.77</td>
<td>74817.15</td>
<td>75477.49</td>
<td></td>
</tr>
<tr>
<td>mean_ccveD</td>
<td>75.10</td>
<td>74.82</td>
<td>75.48</td>
<td></td>
</tr>
<tr>
<td>sd_ccveD</td>
<td>8.90</td>
<td>8.97</td>
<td>9.06</td>
<td></td>
</tr>
<tr>
<td>min_ccveD</td>
<td>60.01</td>
<td>60.02</td>
<td>60.02</td>
<td></td>
</tr>
<tr>
<td>max_ccveD</td>
<td>90.04</td>
<td>90.10</td>
<td>90.03</td>
<td></td>
</tr>
</tbody>
</table>

**TEST STATISTICS**

The measurement results produced by TLSTest are shown in Table 5-2: They consist of any results observed for each of the universal test metrics presented in Section 5.1.1. Moreover, the total number of executed tests \((tscr)\) as well as the mean \((\text{mean}_{tscr})\), standard deviation \((\text{sd}_{tscr})\), min \((\text{min}_{tscr})\) and \((\text{max}_{tscr})\) duration of tests are included. Note that for each TLSTest variant, we only observed a single value for false positive \((\text{fpsD})\) (i.e., \(\text{fpsC}_{FP} = 1\)) and thus we cannot compute average \((\mu_{FP})\), median \((\text{median}_{fpsD,FP})\), and standard deviation \((\text{sd}_{FP})\) for \(TLSTest^{[0,10]}\), \(TLSTest^{[0,30]}\) and \(TLSTest^{[0,60]}\). The corresponding fields of Table 5-2 are marked as *not applicable* (na).

**ACCURACY AND PRECISION OF TLSTEST**

This section presents the results of evaluating the accuracy and precision of \(TLSTest^{[0,10]}\), \(TLSTest^{[0,30]}\) and \(TLSTest^{[0,60]}\).

- **Accuracy and precision based on Basic-Result-Counter \((brC)\):** Table 5-3 shows the results of evaluating \(TLSTest^{[0,10]}\), \(TLSTest^{[0,30]}\) and \(TLSTest^{[0,60]}\) on evaluation measures which are derived from the Basic-Result-Counter \((brC)\) test metric. According to our scenario, the SaaS provider wants to select a configuration of TLSTest which produces the least
number of false positive basic test results \((brC^F)^P\): TLSTest\(^{0.10}\) produced the highest number of \(brC^F\), followed by TLSTest\(^{0.30}\) and TLSTest\(^{0.60}\) (Table 5-2). However, solely relying on the absolute counts of \(brC^F\) is misleading because TLSTest\(^{0.10}\) executed more than twice as many tests TLSTest\(^{0.30}\). Thus, we have to make use of the accuracy and precision measures introduced in Section 5.4.1 which relate \(brC^F\) to the remaining measurement results produced by the test-based measurement technique. These include: Overall accuracy \((oac)^{brC}\), true negative rate \((tnr)^{brC}\), false positive rate \((fpr)^{brC}\), false discovery rate \((fdr)^{brC}\) and positive predictive value \((ppv)^{brC}\).

\(TLSTest^{0.10}\) has the lowest overall accuracy (98.24%) and the lowest true negative rate (97.06%). Further, TLSTest\(^{0.10}\) has the highest false discovery rate (1.55%), followed by TLSTest\(^{0.60}\) (1.46%) and TLSTest\(^{0.30}\) (1.34%). However, the most suitable accuracy measure in context of our scenario is the false positive rate since it captures the ratio between incorrectly passed tests and all test that were expected to fail: TLSTest\(^{0.10}\) has the highest \(fpr\) (2.94%), followed by TLSTest\(^{0.60}\) (2.84%) and TLSTest\(^{0.30}\) (2.64%). As a consequence, the SaaS provider selects TLSTest\(^{0.30}\) if he only relies on the accuracy derived from the \(brC\) test metric.

- **Accuracy and precision based on Failed-Passed-Sequence-Counter \((fpsC)\):** Table 5-4 presents the results of evaluating TLSTest\(^{0.10}\), TLSTest\(^{0.30}\) and TLSTest\(^{0.60}\) based on the universal test metrics Failed-Passed-Sequence-Counter \((fpsC)\). Recall that the SaaS provider within our scenario seeks to configure TLSTest such that it produces the lowest number of false positive results possible. In context of accuracy and precision measures based on the \(fpsC\) test metric, we therefore select the false positive rate \((fpr)^{fpsC}\) and the true negative rate \((tnr)^{fpsC}\) – as defined in Section 5.4.2 – to evaluate the variants of TLSTest since they tell us – out of all events that should have been detected – how many control violation events were correctly detected \((tnr)^{fpsC}\) and how many control violation events were not detected \((fpr)^{fpsC}\).

Despite each of the TLSTest variants only producing a single false positive \(fps\) (see Table 5-2), TLSTest\(^{0.60}\) has the lowest \(fdr\) (0.1%) and the highest \(tnr\) (99.9%) because TLSTest\(^{0.60}\) produced a higher number of true negative \(fps\) (969) than TLSTest\(^{0.10}\) (871) and TLSTest\(^{0.30}\) (893). Hence, if the SaaS provider only draws on the accuracy based on the \(fpsC\) test metric, then he chooses TLSTest\(^{0.60}\).

- **Accuracy and precision based on Failed-Passed-Sequence-Duration \((fpsD)\):** Evaluating TLSTest\(^{0.10}\), TLSTest\(^{0.30}\), and TLSTest\(^{0.60}\) based on the Failed-Passed-Sequence-Duration \((fpsD)\) produces the results shown in Table 5-5. Since only a single value for
false positive $f_{psD}$ (i.e., $f_{psC_{FP}} = 1$, see Table 5-2) for each TLSTest variant has been observed, we cannot calculate mean, median, standard deviation ($sd$) and margin of error ($E^{95\%}$) of $f_{psD_{FP}}$ for TLSTest$^{[0,10]}$, TLSTest$^{[0,30]}$, and TLSTest$^{[0,60]}$. This is indicated by marking the corresponding fields of Table 5-5 as not applicable (na).

Besides choosing a configuration for TLSTest which produces the lowest false positive results, our example SaaS provider prefers the TLSTest variant which as accurately as possible estimates how long it takes the PaaS provider to remedy a detected, vulnerable communication configuration. In other word: The SaaS provider prefers a configuration of TLSTest which most accurately estimates the duration of a correctly detected control violation event.

Figure 5-23 shows three box plots which capture the variation of relative duration error of true negative $f_{ps}$ ($e_{f_{psD_{TN}}}$) for the three TLSTest variants: It is obvious that the relative error each test of TLSTest$^{[0,60]}$ makes when estimating the duration has the highest mean (dashed green line inside the box, 22.96%), median (solid red line inside the box, 20.56%) as well as the highest variability. Further, on average, TLSTest$^{[0,10]}$ produces true negative $f_{ps}$ having the lowest relative error $e_{f_{psD_{TN}}}$ when estimating the duration of a vulnerable communication configuration event (4.56%), followed by TLSTest$^{[0,30]}$ (11.33%). Hence, in context of our scenario, the SaaS provider prefers TLSTest$^{[0,10]}$ since this configuration of TLSTest provides the most accurate estimate of how long it takes the PaaS provider to fix a vulnerable TLS configuration.
Figure 5-23 Relative duration error of fpsD (efePsdTN) of TLSTest[0,10], TLSTest[0,30], and TLSTest[0,60]

- **Accuracy and precision based on cumulative-Failed-Passed-Sequence-Duration (cf psD):** Table 5-6 shows the results of evaluating TLSTest[0,10], TLSTest[0,30], and TLSTest[0,60] using the universal test metric cumulative-Failed-Passed-Sequence-Duration (cfpsD). The results of the total duration error of true negative fps (ecfpsD) show that all evaluated variants of TLSTest underestimate the accumulated duration of vulnerable TLS configuration events. Drawing on ecfpsD, the most accurate result is produced by TLSTest[0,60] (-2232.09 seconds), followed by TLSTest[0,30] (-7476.24 seconds) and TLSTest[0,10] (-9974.22 seconds). However, the accumulated duration of true negative fps is outside the scope of our example scenario since the SaaS provider’s focus lies on correctly detecting temporary vulnerable TLS configurations and estimating their duration. Therefore, the accumulated duration of fps and thus the accumulated error of fps does not affect the decision of the SaaS provider which variant of TLSTest to select.

**CONCLUSION**

The SaaS provider in our example scenario favors TLSTest[0,60] because the accuracy and precision measures efpsC indicate that it has the highest number of correctly detected control violations, that is, true negative fps. One may argue that this conclusion is flawed because TLSTest[0,10] is more accurate in estimating the duration of a vulnerable TLS configuration event (see accuracy and precision measures efpsD). However, we presume that it is more important
to the SaaS provider in our scenario that the continuous test-based measurement technique detects the number of occurrences of control violations most accurately than it is to most accurately estimate the duration of those violations correctly detected. Also, although outside the scope of our example evaluation scenario, comparing the accuracy of the TLSTest variants based on the cumulative error of true negative $fps_{D^{TN}}$ would further foster our conclusion because $TLSTest^{[0.60]}$ produces the lowest value for $ecfps_{D^{TN}}$.

Table 5-2 Summary of test statistics of TLSTest

<table>
<thead>
<tr>
<th>Test statistic</th>
<th>TLSTest</th>
<th>([0,10])</th>
<th>([0,30])</th>
<th>([0,60])</th>
</tr>
</thead>
<tbody>
<tr>
<td>tsrC</td>
<td>34801</td>
<td>13771</td>
<td>7332</td>
<td></td>
</tr>
<tr>
<td>(mean_{tsr})</td>
<td>1.50</td>
<td>1.40</td>
<td>1.38</td>
<td></td>
</tr>
<tr>
<td>(sd_{tsr})</td>
<td>0.59</td>
<td>0.62</td>
<td>0.46</td>
<td></td>
</tr>
<tr>
<td>(min_{tsr})</td>
<td>0.10</td>
<td>0.10</td>
<td>0.10</td>
<td></td>
</tr>
<tr>
<td>(max_{tsr})</td>
<td>19.73</td>
<td>19.39</td>
<td>19.18</td>
<td></td>
</tr>
<tr>
<td>brC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(brC^{TP})</td>
<td>22484</td>
<td>9024</td>
<td>4793</td>
<td></td>
</tr>
<tr>
<td>(brC^{FP})</td>
<td>8</td>
<td>5</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>(brC^{TN})</td>
<td>11585</td>
<td>4504</td>
<td>2410</td>
<td></td>
</tr>
<tr>
<td>(brC^{FN})</td>
<td>260</td>
<td>83</td>
<td>39</td>
<td></td>
</tr>
<tr>
<td>(brC^{PTN})</td>
<td>106</td>
<td>33</td>
<td>18</td>
<td></td>
</tr>
<tr>
<td>(brC^{PFP})</td>
<td>346</td>
<td>118</td>
<td>68</td>
<td></td>
</tr>
<tr>
<td>fpsC</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(fpsC^{TN})</td>
<td>871</td>
<td>893</td>
<td>969</td>
<td></td>
</tr>
<tr>
<td>(fpsC^{FN})</td>
<td>184</td>
<td>110</td>
<td>30</td>
<td></td>
</tr>
<tr>
<td>fpsC&lt;sub&gt;FP&lt;/sub&gt;</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-----------------</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>mean&lt;sub&gt;TN&lt;/sub&gt;</td>
<td>74.78</td>
<td>75.41</td>
<td>75.59</td>
<td></td>
</tr>
<tr>
<td>sd&lt;sub&gt;fpsD,TN&lt;/sub&gt;</td>
<td>9.94</td>
<td>13.93</td>
<td>22.96</td>
<td></td>
</tr>
<tr>
<td>median&lt;sub&gt;fpsD,TN&lt;/sub&gt;</td>
<td>74.55</td>
<td>75.43</td>
<td>75.30</td>
<td></td>
</tr>
<tr>
<td>min&lt;sub&gt;fpsD,TN&lt;/sub&gt;</td>
<td>41.75</td>
<td>39.64</td>
<td>18.20</td>
<td></td>
</tr>
<tr>
<td>max&lt;sub&gt;fpsD,TN&lt;/sub&gt;</td>
<td>97.99</td>
<td>114.76</td>
<td>138.10</td>
<td></td>
</tr>
<tr>
<td>mean&lt;sub_FN&lt;/sub&gt;</td>
<td>52.32</td>
<td>73.46</td>
<td>69.66</td>
<td></td>
</tr>
<tr>
<td>sd&lt;sub_FN&lt;/sub&gt;</td>
<td>32.52</td>
<td>19.29</td>
<td>29.55</td>
<td></td>
</tr>
<tr>
<td>median&lt;sub&gt;fpsD,FN&lt;/sub&gt;</td>
<td>66.17</td>
<td>73.54</td>
<td>65.87</td>
<td></td>
</tr>
<tr>
<td>min&lt;sub&gt;fpsD,FN&lt;/sub&gt;</td>
<td>0.40</td>
<td>13.09</td>
<td>24.27</td>
<td></td>
</tr>
<tr>
<td>max&lt;sub&gt;fpsD,FN&lt;/sub&gt;</td>
<td>96.23</td>
<td>114.18</td>
<td>143.17</td>
<td></td>
</tr>
<tr>
<td>mean&lt;sub&gt;_FP&lt;/sub&gt;</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>sd&lt;sub&gt;_FP&lt;/sub&gt;</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>median&lt;sub&gt;fpsD,FP&lt;/sub&gt;</td>
<td>na</td>
<td>na</td>
<td>na</td>
<td></td>
</tr>
<tr>
<td>min&lt;sub&gt;fpsD,FP&lt;/sub&gt;</td>
<td>87.02</td>
<td>73.02</td>
<td>84.02</td>
<td></td>
</tr>
<tr>
<td>max&lt;sub&gt;fpsD,FP&lt;/sub&gt;</td>
<td>87.02</td>
<td>73.02</td>
<td>84.02</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>fpsD (sec)</th>
<th>TN</th>
<th>65136.55</th>
<th>67340.92</th>
<th>73245.40</th>
</tr>
</thead>
<tbody>
<tr>
<td>FN</td>
<td>9627.13</td>
<td>8080.22</td>
<td>2089.73</td>
<td></td>
</tr>
<tr>
<td>FP</td>
<td>87.02</td>
<td>73.02</td>
<td>84.02</td>
<td></td>
</tr>
</tbody>
</table>
Table 5-3: Evaluation of TLTest to test secure communication configuration of SaaS^OS based on the basic result counter (brC) test metric

<table>
<thead>
<tr>
<th>Test statistic</th>
<th>TLTest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[0,10]</td>
</tr>
<tr>
<td>oac</td>
<td>98.24</td>
</tr>
<tr>
<td>95% oac</td>
<td>0.14</td>
</tr>
<tr>
<td>tnr</td>
<td>97.06</td>
</tr>
<tr>
<td>95% tnr</td>
<td>0.30</td>
</tr>
<tr>
<td>tpr</td>
<td>98.86</td>
</tr>
<tr>
<td>95% tpr</td>
<td>0.14</td>
</tr>
<tr>
<td>fnr</td>
<td>2.22</td>
</tr>
<tr>
<td>95% fnr</td>
<td>0.26</td>
</tr>
<tr>
<td>fpr</td>
<td>2.94</td>
</tr>
<tr>
<td>95% fpr</td>
<td>0.30</td>
</tr>
<tr>
<td>fdr</td>
<td>1.55</td>
</tr>
<tr>
<td>95% fdr</td>
<td>0.16</td>
</tr>
<tr>
<td>ppv</td>
<td>98.45</td>
</tr>
<tr>
<td>95% ppv</td>
<td>0.16</td>
</tr>
<tr>
<td>for</td>
<td>2.18</td>
</tr>
</tbody>
</table>
Table 5-4: Evaluation of TLSTest to test secure communication configuration of SaaS to based on the failed-passed-sequence Counter (fpsC) test metric

<table>
<thead>
<tr>
<th>Test statistic</th>
<th>TLSTest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[0,10]</td>
</tr>
<tr>
<td>\textit{tnr}</td>
<td>99.89</td>
</tr>
<tr>
<td>\textit{E}_{tnr}</td>
<td>0.22</td>
</tr>
<tr>
<td>\textit{fpr}</td>
<td>0.11</td>
</tr>
<tr>
<td>\textit{E}_{fpr}</td>
<td>0.22</td>
</tr>
<tr>
<td>\textit{efpsC} (%)</td>
<td>17.44</td>
</tr>
<tr>
<td>\textit{E}_{efpsC}</td>
<td>2.29</td>
</tr>
<tr>
<td>\textit{npv}</td>
<td>82.56</td>
</tr>
<tr>
<td>\textit{E}_{npv}</td>
<td>2.29</td>
</tr>
</tbody>
</table>

Table 5-5: Evaluation of TLSTest to test secure communication configuration of SaaS to based on the failed-passed-sequence Duration (fpsD) test metric

<table>
<thead>
<tr>
<th>Test statistic</th>
<th>TLSTest</th>
</tr>
</thead>
</table>


<table>
<thead>
<tr>
<th></th>
<th>[0,10]</th>
<th>[0,30]</th>
<th>[0,60]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>efpsD\textsuperscript{TN,(ms)}</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>-52</td>
<td>644</td>
<td>151</td>
</tr>
<tr>
<td>median</td>
<td>254</td>
<td>603</td>
<td>-442</td>
</tr>
<tr>
<td>sd</td>
<td>4508</td>
<td>10465</td>
<td>20991</td>
</tr>
<tr>
<td>min</td>
<td>-22201</td>
<td>-40073</td>
<td>-51054</td>
</tr>
<tr>
<td>max</td>
<td>11552</td>
<td>25906</td>
<td>51510</td>
</tr>
<tr>
<td>$E^{95%}$</td>
<td>300</td>
<td>687</td>
<td>1323</td>
</tr>
<tr>
<td><strong>efpsD\textsuperscript{TN,(%)}_{\text{rel}}</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>4.56</td>
<td>11.33</td>
<td>22.96</td>
</tr>
<tr>
<td>median</td>
<td>3.54</td>
<td>9.62</td>
<td>20.56</td>
</tr>
<tr>
<td>sd</td>
<td>4.15</td>
<td>8.68</td>
<td>16.45</td>
</tr>
<tr>
<td>min</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>max</td>
<td>31.58</td>
<td>48.86</td>
<td>80.83</td>
</tr>
<tr>
<td>$E^{95%}$</td>
<td>0.28</td>
<td>0.57</td>
<td>1.04</td>
</tr>
<tr>
<td><strong>efpsD\textsuperscript{TN,(ms)}_{\text{pre}}</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>4677</td>
<td>10587</td>
<td>21490</td>
</tr>
<tr>
<td>median</td>
<td>4030</td>
<td>9308</td>
<td>19383</td>
</tr>
<tr>
<td>sd</td>
<td>3759</td>
<td>7682</td>
<td>14688</td>
</tr>
<tr>
<td>min</td>
<td>-1582</td>
<td>-774</td>
<td>44</td>
</tr>
<tr>
<td>max</td>
<td>25739</td>
<td>45251</td>
<td>61204</td>
</tr>
<tr>
<td>$E^{95%}$</td>
<td>250</td>
<td>505</td>
<td>925</td>
</tr>
<tr>
<td><strong>efpsD\textsuperscript{TN,(ms)}_{\text{post}}</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>mean</td>
<td>4624</td>
<td>11230</td>
<td>21641</td>
</tr>
<tr>
<td>median</td>
<td>4217</td>
<td>10154</td>
<td>19407</td>
</tr>
</tbody>
</table>
Table 5-6: Evaluation of TLSTest to test secure communication configuration of SaaS OS based on the cumulative failed-passed-sequence Duration (fpsD) test metric

<table>
<thead>
<tr>
<th>Test statistic</th>
<th>TLSTest</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[0,10]</td>
</tr>
<tr>
<td>TN(ms)</td>
<td>-9914223</td>
</tr>
<tr>
<td>ecfpsD&lt;sup&gt;TN&lt;/sup&gt;</td>
<td>TN(%)</td>
</tr>
<tr>
<td>-------------------</td>
<td>-------</td>
</tr>
<tr>
<td>ecfpsD&lt;sup_FN&lt;/sup&gt;</td>
<td>FN(ms)</td>
</tr>
<tr>
<td></td>
<td>FN(%)</td>
</tr>
<tr>
<td>ecfpsD&lt;sup&gt;FP&lt;/sup&gt;</td>
<td>FP(ms)</td>
</tr>
<tr>
<td></td>
<td>FP(%)</td>
</tr>
</tbody>
</table>
6 CONCLUSION

In this deliverable, first a tool chain was presented which implements continuous cloud security audits to support cloud certification. This tool chain draws on existing tools available as background in the EU-SEC project, including:

- **Clouditor**, an example of a continuous test-based measurement technique,
- **STARWatch**, an application to help organizations manage compliance with CSA STAR through self-assessment, as well as
- **Slipstream**, a brokerage service that facilitates deployment of evidence as well as claim storage.

Thereafter, a process was described how to integrate the tool chain with existing cloud services. The steps of this process include:

- Selection of global integration strategy for measurement techniques,
- deployment of tool chain,
- discovery of cloud service,
- derivation of feasible measurement techniques,
- selection of suitable metrics,
- starting execution of measurement techniques, and
- adaption of measurement techniques at operation time.

Finally, an approach was presented which allows to evaluate accuracy and precision of measurement results produced by continuous test-based measurement techniques. To that end, first the universal test metrics $brC$, $fpsC$, $fpsD$, and $cfpsD$ were introduced and it was defined what accuracy and precision mean with regard to cloud service certification. Then, the evaluation process was presented and here the notion of control violation sequences was introduced. Events of these sequences manipulate a cloud service property such that the service does not adhere to one or more controls anymore. These control violation sequences establish the reference values which we treat as the ground truth and which we use to evaluate the accuracy and precision of a technique’s measurement results. Further, at the heart of the evaluation process are so-called evaluation measures which are derived on the basis of the universal the metrics. These measures are derived through comparing events of a control violation sequence with the measurement results computed according to these metrics. These measures permit statements about, e.g., the average error a test-based technique makes when measuring the duration of control violation events. Finally, an example evaluation was
presented where it is shown how - according to some scenario-specific assumptions - a cloud provider can select the most suited configuration for a particular test-based technique.
7 REFERENCES


