EUSEC
EU SECURITY CERTIFICATION

EUROPEAN SECURITY CERTIFICATION FRAMEWORK

D3.3 ARCHITECTURE AND TOOLS FOR EVIDENCE STORAGE

V 1.0

PROJECT NUMBER: 731845

PROJECT TITLE: EU-SEC

DUE DATE:31/12/2017	DELIVERY DATE: 27/12/2017
AUTHOR:	PARTNERS CONTRIBUTED: CaixaBank, Fraunhofer AISEC, SI-MPA, SixSq
DISSEMINATION LEVEL:* PU	NATURE OF THE DELIVERABLE:** R
INTERNAL REVIEWERS: CSA, MF SR	

*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

The EU-SEC project proposes continuous certification as an enhancement of the current manual certification procedures by incorporating automated and continuous workflows for collecting and evaluating evidences. The existing tools providing support for the continuous security audits of cloud services operate based on monitoring- and test-based techniques that produce evidences.

This deliverable proposes the design and implementation of an architecture which ensures trustworthy, reliable and performant management of evidences. This includes storage of evidences as well as all the interfaces, data formats and protocols required to provide a seamless and generic transport of data from the evidence producer to the storage element, and from the storage element to data consumers and CSP end-users.

Building on the content from past deliverables, evidences are seen as cloud infrastructure resources and are adopted by an infrastructure management model and protocol described by the CIMI specification. In agreement with this specification, the desired data structure for the produced evidence is fine-tuned and the mechanisms for user authentication and authorization are here introduced.

In order to provide a scalable and analytical approach for the storage of evidence, ElasticSearch will be used, introducing a document based evidence management instead of the traditional relational databases.

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.

ABBREVIATIONS

EU-SEC	European Security Certification Framework
DBMS	Database Management System
JSON	JavaScript Object Notation
DSL	Domain Specific Language
CSP	Cloud Service Provider
CIA	Confidentiality, Integrity, Availability
CIMI	Cloud Infrastructure Management Interface

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

1 INTRODUCTION

Many of the existing certification procedures nowadays are manual (requiring human intervention). When applied to Cloud Service Providers, these procedures are still sub-optimal when it comes to efficiency and effectiveness of the used security certification techniques. While other work packages within the EU-SEC project are focusing on the collections of requirements and definition of controls for improving the existing security certification methods, this work package's goal is to design methods and implement components for a production deployment of cloud services security audits, in a continuous manner, so that the existing manual certification procedures can be incorporated with ongoing research approaches to automatically generate and evaluate evidences. Apart from all of the above, evidences have to be produced in a particular way and following strict security requirements, because they have to preserve their independence of the entire system in the sense that they are the material from which audit and certification is based, and therefore the credibility of the entire system is based on the credibility of the evidences.

In deliverable (D3.2), tools and a Domain Specific Language have been defined that alongside monitoring-based and test-based techniques, can be used for automating the generation of security certification evidence for CSPs, furthermore allowing the processing of evidence based on measurements which return the necessary output for performing the control objective evaluation and therefore issue claims based on the existing SLOs or SQOs. This workflow provides the necessary foundation for an automatic and continuous auditing process.

Evidences can be used for several reasons, and it will be necessary to cover different use cases, such as:

1- Evidences could be used by third parties such as auditors or regulator bodies to verify that the Cloud service is achieving the audit and certification criteria
2- Evidences could be used by the CSP as contrast to their own indicators and verify that there is not any incongruence. CSP can also use evidences prove and report that the Service is actually achieving the level of privacy and security. Due to the fact that evidences are produced in a secure and independent way CSP can reinforce their reports from an independent point of view.
3- Evidences can be used by the CSC to monitor the Service, with two main advantages, on one hand evidences are produced directly from the service and in principle can't be manipulated by the CSP, and on the other hand, evidences can be integrated in the infrastructure of the CSC and therefore the CSC can aggregate and correlate evidences with internal information in an integrated way.

To be able to assure that evidences can be trusted, in general they have to follow the three basic requirements of security, known as CIA (Confidentiality, Integrity and Availability) apart from the non-repudiation. These security requirements are further developed in chapter 2.4.3 security requirements for trustworthy evidence production and storage

1.1 OBJECTIVES

The purpose of this deliverable is to build on top of the continuous auditing architecture defined in D3.2, adding the ability to both store and manage evidence through a standard interface. The proposed management solution shall provide secure data transfer from the evidence producer to the storage element and from the latter to an end-user, an appropriate evidence storage technology that eases the search, filtering and analytical processing of the stored evidence, and different means of retrieving and even visualizing data, both through an API and a web interface.

The proposed evidence management architecture shall take into account privacy aspects, making sure evidence cannot be compromised both through direct access to the storage element or anonymous and unauthorized access to data.

This document also introduces the concept of evidence cataloguing, whereby a 3rd party tool can make use of the stored evidence to generate a catalogue of cloud service offers from which end-users can search and perform tailored queries according to certain security requirements, thus optimizing the selection of a cloud provider prior to the actual deployment of services and applications in the infrastructure. This 3rd party tool shall not only offer the evidence catalogue for different CSPs, seamlessly, but also provide the ability to ease the setup of the continuous auditing process throughout different cloud providers, thus minimizing boilerplate code during deployment.

Evidences should be comply with all the legal requirements to be used as a proof in case of trial, so they have to guarantee the chain of custody.

The outcome of this deliverable shall serve as a foundation for the pilot phase addressed in work package 5, where the different project tools and defined architectures will be implemented and integrated.

1.2 DOCUMENT STRUCTURE

This deliverable first starts by addressing the continuous auditing and certification architecture already defined in deliverable D3.2, providing a summary on how evidence is produced and which kind of external factors and assessments can influence not only the generation and trustworthiness of the evidence, but also its management. The existing data format and

structures in which evidence are produced (*testResult*) are exemplified. Afterwards, in section 3, the actual backend for storing and handling evidence is described, detailing the interface to be used to manage evidence in a secure way, as well as a document based search engine for storing (and possibly visualize) evidence and the respective attribute mapping that should be followed to maintain consistency amongst *testResults*.

2 CONTINUOUS AUDITING AND CERTIFICATION ARCHITECTURE

In this section, the production, trustworthiness and representation of automatically produced evidence is described. To that end, section 2.1 describes the role of test-based evidence production in context with the other concepts which are needed to allow for continuous security audits of cloud services. Thereafter, section 2.2 und 2.3 outline challenges with regard to the quality and security properties of evidence. These challenges may affect the trustworthiness of the evidence with regard to, e.g. certification authorities, service customers or service providers. Lastly, section 2.5 introduces a data structure how to represent an instance of evidence within the evidence store.

2.1 EVIDENCE PRODUCTION AS PART OF CONTINUOUS SECURITY AUDITS

In order to understand the concept of the evidence storage in the context of continuous security audits, recall how evidence is produced and processed as introduced in Section 3 of Deliverable 2.2 as well as in Section 1.1.1 of Deliverable 3.2. Figure 1 provides an overview how the different concepts when implemented by a concrete tool chain support continuous security audits: Evidence production techniques provide, e.g. by using tests, some form of evidence, e.g. supported TLS cipher suites of a cloud service's public endpoint (Step 1).
There are two approaches to continuous evidence production (Cimato, 2013):

- *Monitoring-based evidence production:* These techniques use monitoring data as evidence which is produced during productive operation of a cloud-service (Stephanow P. a., 2015). Two major types of monitoring-based evidence production techniques can be distinguished: The first group consists of methods proposed by current research (e.g., Krotsiani et al. (Krotsiani, 2013), Schiffmann et al. (Schiffman, 2013)) which are specifically crafted to produce evidence to check whether particular properties of a

cloud service are satisfied, e.g. integrity of cloud service components (Schiffman, 2013) and correctness of non-repudiation protocols used by cloud services (Krotsiani, 2013). Those methods require implementing additional monitoring services which are not needed for operational monitoring of the cloud service. The second group of monitoring-based evidence production techniques consists of existing monitoring services and tools which are used to operate the infrastructure of a cloud service, e.g., Nagios or Ganglia. The data produced by these monitoring tools can also be used as evidence to check a cloud service's properties such as availability (Stephanow P. a., 2015). Additionally, data produced by tools which aims to detect intrusions such as Snort, Bro, or OSSEC can serve as evidence (Stephanow P. a., 2015) (Stephanow P. a., 2015).

- *Test-based evidence production:* Similar to monitoring-based techniques, test-based evidence production also collects evidence while a cloud-service is productively operating. Different to monitoring-based techniques, however, test-based techniques do not passively monitor operations of a cloud service but actively interact with it through tests. Thus test-based methods produce evidence by controlling some input to the cloud service, usually during productive operation, e.g. calling a cloud service's RESTful API (Cimato, 2013) (Stephanow P. a., 2017).

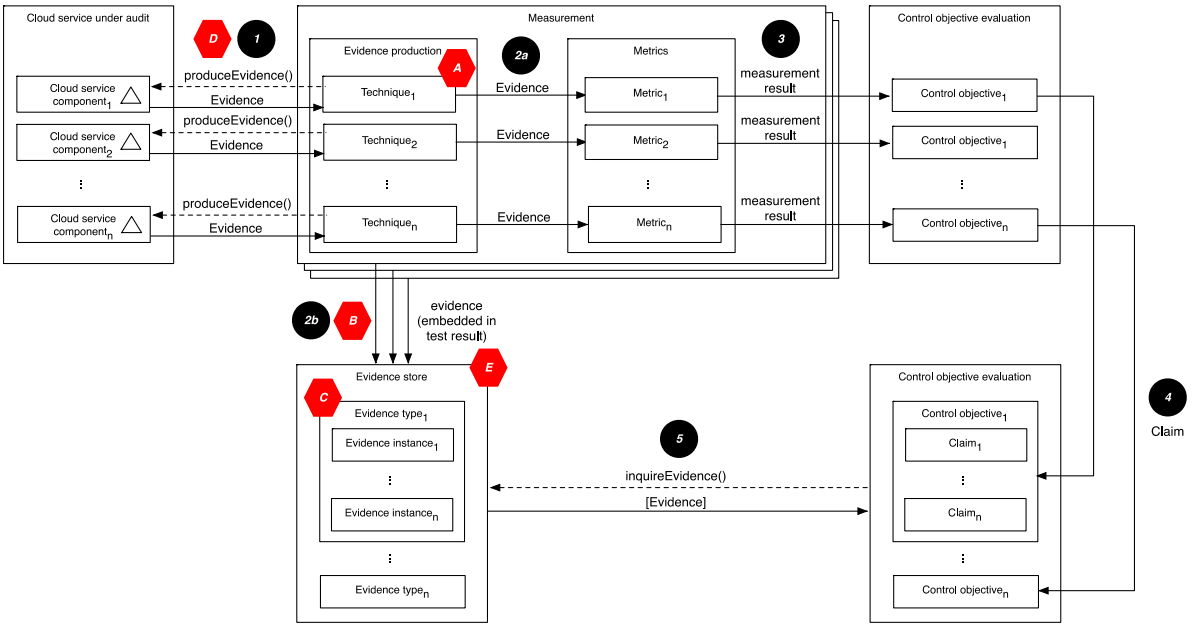

Figure 1 Continuous security audits (including potential target points of attack)

In the course of the EUSEC project, the focus lies on test-based evidence production; monitoring-based is *not* considered. As Figure 1 indicates, each produced instance of evidence is forwarded to the *evidence store* (Step 2). More specifically, a *test result* is sent to the evidence

store. Recall that – as described in Section 4.1.5 of Deliverable 3.2 – a test result contains an instance of evidence. Only parts of a *singular* test result are considered *evidence,* whereas the entire test result already implies that a decision has been made based on the information observed during the test's execution. Thus, any information which serves as input to a well-defined test oracle which is part of test cases constitutes evidence.
Instances of evidence are then further processed by a *metric*, i.e. a function which takes *evidence* as input and outputs *measurement results* (Step 2a). This refines and operationalizes the general definition of the term *metric* provided by Deliverable 1.4 where a metric is a "specified process for obtaining a value".

A *measurement technique* consists of at least one evidence production technique and one metric. In context of the TLS cipher suite example, a concrete metric may inspect the list of supported cipher suites and check whether it only contains those of a predefined whitelist which are considered secure. A measurement result produced by that metric either indicates that all supported cipher suites are secure (*isSecure*) or are not secure (*isNotSecure*).
After having been produced, *measurement results* are forwarded to *control objective evaluation* (Step 3). Satisfaction of a control objective, again, can be described as a function which takes a *measurement result* as input and outputs a *claim*, that is, a result indicating whether a control objective holds at some point in time.

Once a claim has been produced, it is then forwarded to the *claim store* (Step 4). In case a dispute arises, e.g. between a service customer and the service provider, then claim store can inquire the evidence which has been used to produce the claim.

- *Based on controls:* In this case, evidences are defined directly from the control. It should be possible to define which kind of evidences are needed to verify Control is correctly applied and taking into account that they can be used to provide proof that the Control is correctly applied, therefore starting from a particular Control, registries, logs and documents should be produced in order to demonstrate the level of applicability of the Control.

2.2 ACCURATE EVIDENCE PRODUCTION

Evidence produced by some evidence production technique forms the atom upon which the reasoning about particular cloud service properties is based. Satisfaction of these properties, in turn, determines whether a control objective holds.

Inaccurate evidence undermines both the cloud service providers' and the customers' trust: On the one hand, evidence that *incorrectly* leads to the conclusion that a control objective is satisfied erodes the customer's trust. On the other hand, cloud service providers will dispute evidence that incorrectly suggest control objectives are not fulfilled. Therefore, it is essential to

evaluate the accuracy and precision of evidence production techniques, that is, how close is produced evidence to the truth?

Consider the control *TVM-02 Threat and Vulnerability Management* of CSA's Cloud Control Matrix (CCM) which reads

> *"Policies and procedures shall be established, and supporting business processes and technical measures implemented, for timely detection of vulnerabilities within organizationally-owned or managed (physical and virtual) applications and infrastructure network and system components, applying a risk-based model for prioritizing remediation through change-controlled, vender-supplied patches, configuration changes, or secure software development for the organization's own software [..]."*

One possibility to continuously produce evidence supporting validation of this control is to execute a vulnerability scanner every ten minutes and then use the output of this scanner as evidence to check whether vulnerabilities are found, and if found, whether they it has been detected in time. The question is now whether the evidence production technique makes mistakes by, e.g., by incorrectly indicating that the cloud service under test has no vulnerabilities while it actually has. In this case, it is unclear to what extend the produced evidence can be used to determine the test result as well as compute more complex measurement results which are then used to check if control objective is satisfied. Does, e.g., the vulnerability scanner always miss to detect a particular vulnerability or merely occasionally? In the next example, we assume that the evidence production technique used to test for security vulnerabilities only produces correct evidence. When inspecting control TVM-02, it becomes apparent that it not only requires detecting security vulnerabilities but also demands remedy within specific period of time. In context of such temporal constraints, further errors may occur when – based on metrics using the evidence – estimating the duration of detected vulnerabilities.

Addressing the above challenges, Deliverable 3.4 will introduce a method how to experimentally evaluate the accuracy and precision of continuous test-based measurement techniques. This method allows comparing alternative test-based measurement techniques as well as comparing alternative configurations of test-based techniques. Furthermore, it permits to infer general conclusions about the accuracy of a specific test-based measurement technique.

It should be possible to relate Controls with evidences, that is to say, starting from a Control, to know the set of evidences which are related to this Control, and the other way round, from each evidence to know to which Control is related with.

Taking into account what has been said previously regarding the validity of the evidences in a legal process, security of the evidences should be possible to demonstrate, as well as the chain of custody and their metadata, such as timestamp.

2.3 TRUSTWORTHY EVIDENCE PRODUCTION AND STORAGE

This section describes high-level requirements for trustworthy evidence production and storage. To that end, examples of attack scenarios on evidence production and storage are outlined. Those scenarios are derived from the high-level description of the concepts involved in continuous security audits shown in Figure 1. Then, given these attack scenarios, security requirements for trustworthy evidence production and storage are derived.

It is important to note at this point that the requirements introduced in this section are neither complete nor sufficiently concrete to derive a (risk-based) security model for evidence stores. Deriving such a security model involves, among others, defining a realistic attacker, having detailed architecture of the involved systems available, conducting security tests (e.g., penetration tests) to determine the probability of a successful attack and estimating of damages resulting from, e.g., modified instances of evidence. It is obvious that these conditions depend on (and thus vary with) the concrete scenario in which a particular cloud service should be continuously audited. Additionally, there is a multitude of variants feasible when implementing a concrete tool chain to support continuous security audits. Therefore, a risk-based security model has to be derived in context of a concrete deployment of the tool chain with a particular cloud service. The process of risk-based deployment of the tool chain will be described in detail as part of Deliverable 3.4.

2.4 AFFECTING EVIDENCE MANAGEMENT

2.4.1 POTENTIAL THREATS

The audit evidences must be provided and available in the automatic way at the point when the continuous auditing happens. We identified several influences (threats) that might impact the collection, management and use of the evidences. This can be structured in next areas:

- Changes of the requirements for the cloud services
- Interruption of cloud services
- Security, safety and accuracy of tools supporting continuous auditing
- Documentation of evidences storage and used tools
- Use of evidences.

Table 1. Threats impact and measurement

Changes of the requirements for the cloud services		
Threats	**Impact**	**Measure**

Frequent changes in the legislation or in the requirements of the CSC	Tool doesn't provide the functionalities capable to provide evidences compliant with the new requirements	- Changes of SLA
Diverse required retention period and frequency of providing and recording evidences by legislation or individual CSC	Impact on the storage capacities and technologies	- Separate storage - Multi-tenant cloud service environment - Scalable storage capacities - Location of the data (inside specific geographical coverage)
Interruption of cloud service		
Threats	**Impact**	**Measure**
External threats will originate from sources outside of the organization and its network of partners which can interrupt the cloud service. Examples include criminal groups, lone hackers, former employees etc. Typical examples include: • Phishing: phishing as a means to install persistent malware with stolen credentials • Human element: social engineering, financial pretexting, digital extortion, insider threat, partner misuse • Conduit devices: peripheral tampering, USB infection, hacktivist attack, rogue connection, logic switch	Typical threats could have impact on integrity, availability and confidentiality of cloud services.	The tool shall provide all aspects of information security (integrity, availability and confidentiality) and shall be complied also with Data Protection Act. To eliminate the impacts of external threats the tool shall be used (manual or automatic) together with other tools or measures which detect cyber attacks (SOC).

• Configuration exploitation: backdoor access, SQL injection, CMS compromise, DNS tunneling • Malicious software: data ransomware, sophisticated malware, credential theft • Data breaches: involved weak, default or stolen passwords Web applications: The great complexity of the infrastructure makes web application servers a target for attackers. Web sites are not static pages anymore, they are highly interactive and more complex.		
Security, safety and accuracy of tools supporting continuous auditing		
Threats	**Impact**	**Measure**
Disruption of evidence production and storage	Evidence production or evidence storage may not be operational for some time which leads to gaps in the continuously produced evidence.	The tool for collecting and storage of evidences, together with communication lines, shall provide a high level of availability with the option of real time monitoring. All intentional interruptions shall be planned and documented (upgrades of tools, vulnerability checks, business continuity plans).
Inaccurate evidence production	The tooling which implements continuous security audits may be incorrectly configured or erroneous resulting in the production of incorrect instances of evidence.	- Trainings, presentations, available documentation for the use of the tool - Change control process - Analytics tools for assessment of evidences
Vulnerability of tools	Vulnerabilities of tools may lead to disrupted or	- regularly test and check the tool

	modified evidence production which render produced instances of evidence unusable for continuous security audits	- Eliminate bugs by updating the tools with new versions - Controlled development of the tool
Documentation of evidences and tools		
Threats	**Impact**	**Measure**
- Lack of documentation - Outdated documentation	- Incorrect interpretations of collected evidences - Evidences are useless - Unknown significance of data	- Up-to-date documentation of tools (specifications, configuration, versioning) and evidences - Requirement: 16.1.7 / ISO27017 and A.16.1.7 / ISO27001 (procedures for the identification, collection, acquisition and preservation of information, which can serve as evidence - *Examples from D1.2 combined requirements)*
Use of evidences		
Threats	**Impact**	**Measure**
Different interpretation of evidences	Different interpretations leading to different auditing assessments/conclusions	- Trainings, presentations, available documentation
Auditors' skepticism to trust the evidences provided by tools (low level of trustfulness, validity and usefulness)	- Available evidences are not used in audit process - Lower level of audit automation	- Trainings, presentations, available documentation - Official recognition of continuous auditing tools and evidences in auditing process by certification bodies

2.4.2 ILLUSTRATIVE ATTACK SCENARIOS ON EVIDENCE PRODUCTION AND STORAGE

This section presents a set of exemplary, high-level attack scenarios which are derived from the continuous security audit concepts shown in FIGURE *1* which were introduced in Section 2.1.

ATTACKING THE INTEGRITY OF EVIDENCE PRODUCTION AND STORAGE

The integrity of evidence can be subjected to an attack, that is, someone intentionally tries to modify the data which an instance of evidence contains. Such unauthorized alteration of evidence may be attempted at different points during evidence production and storage. Hereafter, some exemplary high-level scenarios are described which outline how an attacker may compromise the integrity of evidence and discuss consequences.

To begin with, an attacker can try to modify the application which is used to produce the evidence (see entry point *A* of Figure 1), i.e. the implementation of the evidence production technique (which, as point out in Section 2.1, conceptually is part of the measurement technique). For example, an evidence production technique which parses the configuration file of a web server to discover supported TLS cipher suites may be altered by an attacker to *not* include those cipher suites in the produced evidence which are known to contain vulnerabilities. A measurement conducted on the basis of this evidence will thus *not* indicate the endpoint actually supports vulnerable TLS configurations. Furthermore, the evidence persisted in the evidence store will *not* reflect the actual status of the cloud service under audit. If successful, this type of attack is particularly perfidious since even if a customer later discovers by herself that claims produced by the continuous audits do not accurately reflect reality, then inquiring the instances of evidence which led to the computation of the claim will be consistent with the claim, thereby falsely rejecting the objection brought forward by the customer. However, there is no means by which the customer can prove that her objection to the produced claims is in fact valid.

As another example, consider an attack to modify instances of evidence when they are transferred from the evidence production technique to the evidence store (see entry point *B* of Figure 1). In this scenario, the implementation of the evidence production technique remains unaltered and instances of evidence are altered only after having been produced accurately. This implies that measurement results computed on the basis of this evidence are also accurate, that is, they contain correct information about whether a cloud service has a certain property or not. However, since the instances of evidence are manipulated during transit from the evidence production technique (or measurement technique) to the evidence store, these instances which are persisted by the evidence store are *not* trustworthy. When considering the retrieval of evidence in case there is a dispute over a claim, such modified evidence may incorrectly contradict claims: Although the claims are actually correct, because they have been

computed using unaltered measurement results, the altered and thus incorrect instances of evidence persisted in the evidence store suggest otherwise.

Lastly, an attacker may try to gain unauthorized access to the evidence store in order to manipulate stored evidence (see target point *C* of Figure 1). In this case, the evidence production techniques as well as transfer between the evidence production and evidence store have not been compromised. Similar to the scenario of manipulating evidence during transit, also in this scenario measurement results are computed based on accurate instances of evidence. Yet, instances of evidence, which have been altered through unauthorized access to the evidence store are *not* trustworthy. Therefore, when retrieving evidence used to compute a claim, then it is unclear whether evidence contradictory to the claim is trustworthy.

ATTACKING THE AVAILABILITY OF EVIDENCE PRODUCTION AND STORAGE

The availability of evidence can be the target of an attack, i.e. an attacker tries to disrupt evidence production or evidence storage. The next two paragraphs describe how an attacker may disrupt production and storage of evidence as well as points out the potential consequences of such types of attacks.

An attacker may disrupt the production of evidence at the very moment at which an evidence production technique attempts to produce instances of evidence (see target point *D* of Figure 1). Recall the example of an evidence production technique which aims to identify all TLS cipher suites supported by a cloud service's endpoint. Let's consider that the supported TLS cipher suites are produced by the technique through actively connecting to the endpoint and conducting TLS handshakes. If an attacker is able to disrupt these communication attempts, then the evidence production technique cannot obtain the desired evidence. As a result, due to the lack of evidence, no measurement results can be computed and thus the evaluation as to whether a cloud service only uses strong cipher suites at a certain point in time is not feasible. Naturally, since no instance of evidence is produced, none are forwarded and persisted in the evidence store. It is obvious that a successful attack of this type disrupts the entire continuous security audit process. Therefore, as long as this attack persists, no claims are produced because they draw on measurement results whose computation is not feasible due to the unavailable instances of evidence.

In a different scenario, an attacker may not be able to directly disrupt evidence production technique but to disrupt the insertion of instances of evidence into the evidence store (see target point *B* of Figure 1). In this case, the evidence production techniques work as expected. Only after an instance of evidence has been correctly produced, its subsequent persistence in the evidence store is disrupted. There are multiple attack vectors which may prevent the insertion of an instance of evidence into the evidence store. For example, the attacker may access the evidence during transit and alter it in such a way that it is discarded as invalid input when provided to the evidence store. Note that since the evidence production technique works correctly, measurement results can be computed, thus allowing to produce claims at a certain point in time. Yet, as pointed out above, the corresponding instances of evidence used to derive the claim are not persisted in the evidence store. This implies that it is not feasible to retrieve

the evidence which was used to compute a particular claim. Therefore, in case of a dispute, a produced claim cannot be substantiated with evidence.

Lastly, an attacker can focus on disrupting the execution of the evidence store itself (see target point *E* of Figure 1). Similar to the case above, evidence production as well as computing measurements results and claims will function as expected. However, instances of evidence cannot be persisted at the evidence store because the attacker managed to force it into a state of service denial. A possible attack vector may leverage vulnerabilities in the authentication procedure of the evidence store, or the exhaustion of the evidence store's resources (e.g., the VM it is running on), rendering it unable to process other requests (such as storing instances of evidence provided by the evidence production technique). In case of a successful attack, no instances of evidence can be persisted at the evidence store and thus produced claims cannot be verified through retrieving the corresponding evidence.

2.4.3 SECURITY REQUIREMENTS FOR TRUSTWORTHY EVIDENCE PRODUCTION AND STORAGE

Hereafter, a set of general security requirements are described which are derived on the basis of the exemplary attack scenarios outlined in the previous paragraphs. These requirements are sorted by the groups' governance, integrity, availability, authenticity, authorization, non-repudiation, and consistency:

- Governance
  - **Gov01**: Applications which implement measurement techniques (part of which are evidence production techniques) as well as evidence stores have to follow a suitable security model.
  - **Gov02**: The security properties of the measurement technique and the evidence store have to be tested and verified (through different suitable means of review and testing) initially (i.e., at time of first deployment) as well as incrementally on any change made to the applications, their configuration or their deployment environments.
  - **Gov03**: Applications implementing measurement techniques (including evidence production techniques) and evidence stores, their configurations as well as their deployment environments have to conform to their last accredited version.
- Confidentiality
  - **Conf01:** Instances of evidence persisted at evidence stores shall only be disclosed to authorized entities.
  - **Conf02**: Instances of evidence forwarded by measurement techniques to evidence stores shall not be disclosed to any entity during transit.
  - **Conf03**: Instances of evidence produced by measurement techniques shall only be disclosed to authorized entities after production, before forwarding instances to an evidence store.

    - **Conf04**: Instances of evidence shall not be persisted by the measurement techniques and shall be irrecoverably deleted immediately after having been forwarded to the evidence store.
- Integrity
    - **Int01**: Applications which implement measurement techniques and evidence stores, their configurations and their deployment environments may only be modified before initial deployment or during operation by authorized entities.
    - **Int02**: Instances of evidence provided by evidence production techniques shall only be modified by authorized entities after production.
- Availability
    - **Avail01**: Communication between measurement techniques and evidence store has to be feasible at all times (with some level of confidence).
    - **Avail02**: Deployed evidence production techniques have to be able to produce instances of evidence at all times (with some level of confidence).
    - **Avail03**: Forwarding produced instances of evidence from measurement techniques to the evidence stores has to be always feasible (with some level of confidence).
- Authenticity
    - **Authn01**: Measurement techniques and evidence stores have to mutually authenticate each other before instances of evidence are forwarded from a measurement technique to an evidence store.
    - **Authn02**: Parties which request access to the evidence store must authenticate themselves with the evidence store.
- Authorization
    - **Authz01**: Only measurement techniques with proper authorization can store instances of evidence at an evidence store.
    - **Authz02**: Only authorized entities can access evidence stores, modify their configurations and their deployment environments.
    - **Authz03**: Authorized entities can only read those instances of evidence persisted in the evidence store for which they have proper permissions.
    - **Authz04**: Authorized Entities can only modify those instances of evidence persisted in the evidence store for which they have proper permissions.
    - **Authz05**: Only authorized entities can access measurement techniques, modify their configurations and their deployment environments.
- Non-repudiation
    - **NonRep01**: Modifications of measurement techniques, their configurations or their deployment environments have to be documented without exception and in a non-repudiable manner.
    - **NonRep02**: Modifications of instances of evidence after production through accessing the measurement technique have to documented without exception and in a non-repudiable manner.
    - **NonRep03**: Access to the evidence store has to be documented without exception where the entity responsible for the transaction can be rigorously identified (i.e., it is possible to formally prove that an entity has accessed the evidence store).

- Consistency
  - **Consist01**: Measurement results shall only be computed based on instances of evidence that has been persisted at the evidence store (with some level of confidence).

2.5 EVIDENCE FORMAT

As mentioned above in Figure 1 Continuous security audits (including potential target points of attack), evidences will be embedded in the test results coming out of the test-based evidence production engine. These results are here addressed as raw data and in practice they are represented in a JSON format, which allows for a direct injection into the evidence storage element addressed in Section 3.2.

Two real test result samples are exemplified in Appendix A. Even though the evidence structure might evolve alongside with the architecture for continuous auditing and evidence management, the key values to retain from these records are:

TestSuiteResult

- *TestSuiteResult.**_id***: ID given by the evidence production engine DB
- *TestSuiteResult.**className***: given by same DB as above
- *TestSuiteResult.**testId***: unique identifier for the test instance which produced the test results
- *TestSuiteResult.**startTime***: point in time when the test suite execution was triggered (a single test execution within a continuous test always corresponds to executing a test suite)
- *TestSuiteResult.**endTime***: point in time when the test suite execution completed, i.e. all test cases bound to a test suite completed (a test suite always binds at least one test case)
- *TestSuiteResult.**passed***: the result of the test suite execution (a test suite only passes if all contained test cases pass)

TestSuiteResult.source

- *source.**interval***: 2-tuple from which waiting time after completion of last test suite execution is chosen
- *source.**randomized***: waiting time after completion is chosen either randomly (true) or only the first bound of the interval is used for static waiting time after completion if set to false
- *source.**label***: descriptive name for a test suite
- *source.**offset***: additional fixed offset added to waiting time after completion
- *source.**iteration***: maximum number of successive executions of the test suite (if set to -1, then iterations are unlimited)

TestSuiteResult.testCaseResults

- *testCaseResults*: array of results return by each test case which is executed as part of the test
- *testCaseResults.**[i]._id***: test case ID given by the DB
- *testCaseResults.**[i].details***: detailed results which a test case returns and which can be used to compute a metric (measurement result), e.g. '*accessiblePorts*' in the case of PortTest, or '*hasHeartBleed*' in the case of TLSTest (see Appendix A)
- *testCaseResults.**[i].source***: contains the specification of the test case that produced the testCaseResult.[i]
- *testCaseResults.**[i].source.className***: given by the DB
- *testCaseResults.**[i].source.[testOracle]***: list that specifies the expected values which are used to determine whether a test passes or fails, e.g. in the PortTest, this key is named '*expectedPorts*' while in TLSTest, it is called '*expectedBlacklist*' (see Appendix A)
- *testCaseResults.**[i].source.name***: descriptive name of the test case
- *testCaseResults.**[i].source.order***: priority (1 is highest) of test case execution, i.e. the order in which a test suite executes test cases
- *testCaseResults.**[i].source.timeout***: maximum time a test case waits until a result is returned
- *testCaseResults.**[i].source.toolName***: name of an external tool if applicable, e.g. for TLSTest, it is *sslyze*, for PortTest, it is *nmap* (see Appendix A)
- *testCaseResults.**[i].source.startTime***: point in time when the test case execution was triggered
- *testCaseResults.**[i].source.endTime***: point in time which the test case execution completed
- *testCaseResults.**[i].source.passed***: the result of the test case execution

Note: the fields hereafter are specific to test case implementations, that is, there are a couple of fields in '*testCaseResults.[i].source*' which depend on the test case:

- *testCaseResults.**[i].source.host***: target IP or hostname of the service endpoint which the test points to (not always applicable, some cases may require a URL to address a specific resource, e.g. if a particular API is called by the test)
- *testCaseResults.**[i].source.?***: Miscellaneous test case specific fields, e.g. '*runAsRoot*' if external tooling requires root privileges; in the TLSTest, the field '*sslPort*' specifies which port is expected to support TLS (see Appendix A), etc.

3 EVIDENCE MANAGEMENT

With basis on deliverable D3.2 and the architecture presented in Figure 1, the continuous auditing process is hereby simplified in Figure 2.

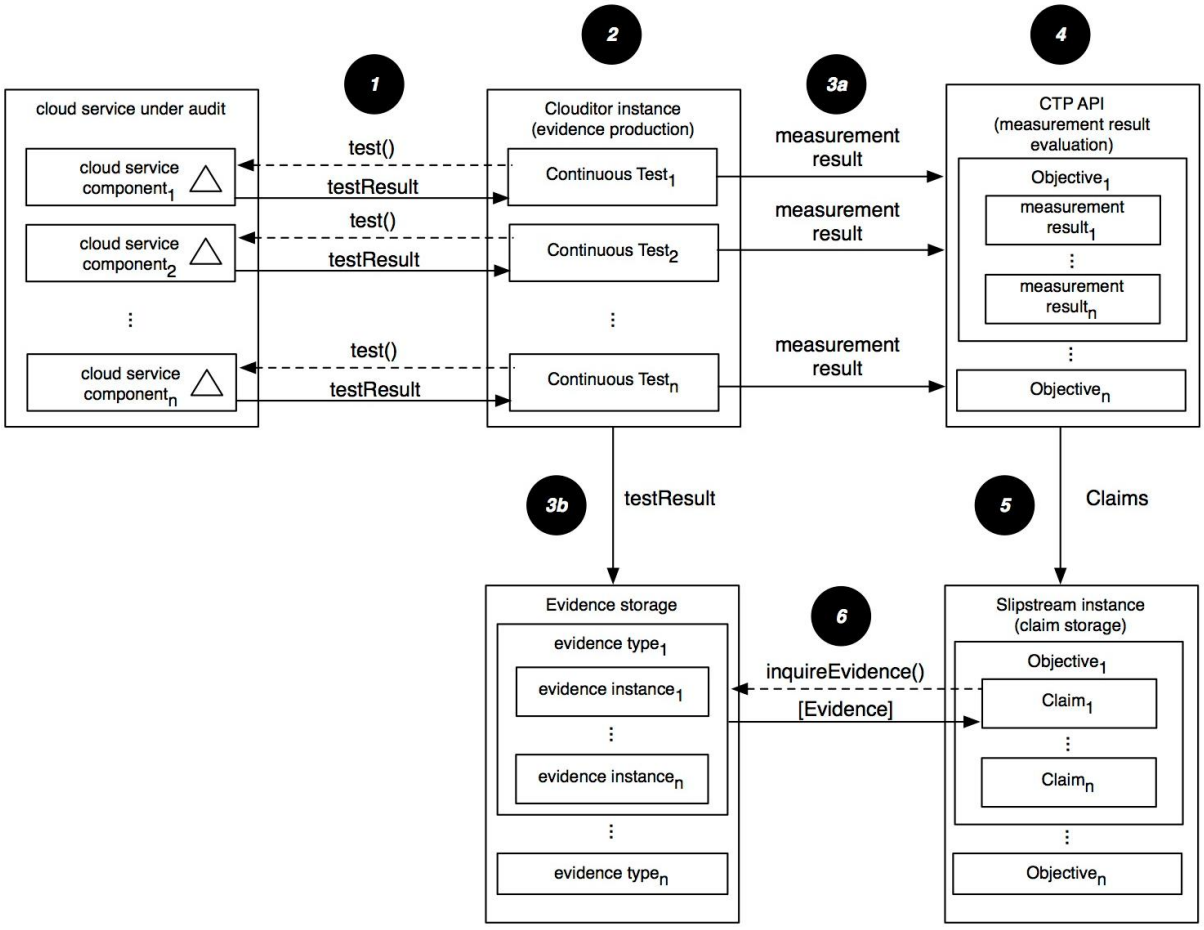

Figure 2 - Continuous auditing process

The evidences are enclosed by the *testResults* produced by Clouditor. Each numbered circle in the diagram represents the execution step and order in which evidences are managed. These have already been described in section 2.1. The only noticeable differences here are the introduction of SlipStream[1] in between step 5 and 6, acting as a cataloguing service for the continuous auditing claims and evidences, which ultimately will be presented to end-users to allow an optimized selection of a cloud provider.

[1] http://sixsq.com/products/slipstream/

3.1 INTERFACES

To achieve a modular architecture whereby evidences can be managed independently of the evidence generator and storage technology (producer and consumer), an interface is hereby proposed, which can be deployed in a standalone mode (a micro-service), can easily be adapted to different surrounding technologies and which sits alongside the storage element as shown in the Figure 3 snippet below.

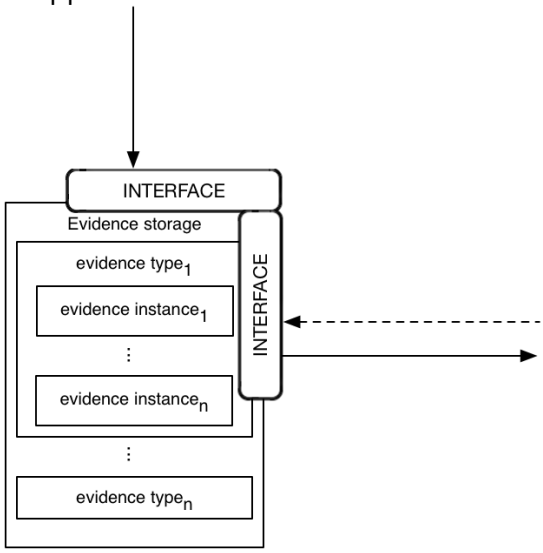

Figure 3 - Snippet of the full architecture, including the evidence management interface

Since evidences can be seen as parameters of the Cloud Service Provider, they can be represented as infrastructure resources thus fitting well with the Cloud Infrastructure Management Interface (CIMI) model and RESTful HTTP-based protocol for management interactions, as described by DMTF[2].

CIMI provides a standard for the management of resources within an infrastructure. For the EU-SEC framework, even though the evidences will be gathered by 3rd party tools, their content will still be CSP dependant and can therefore be interpreted as a derived infrastructure resource. All evidences shall be modelled and represented in JSON (and also XML if possible) according to the CIMI specification. These will be identified by URIs, whereby each evidence representation shall have a globally unique ID attribute of type URI which acts as a reference to itself.

Beside its ease of use and wide industry support, this high-level interface provides:

- consistent resource management patterns, making it easy to develop small, lightweight and infrastructure agnostic clients;
- auto discovery of new resources without changing clients, enabling dynamic evolution of the platform;
- standard mechanism for referencing other resources;

[2] http://www.dmtf.org/sites/default/files/standards/documents/DSP0263_2.0.0.pdf

- flexibility to cover a wider range of resources than strictly those related to cloud infrastructure management.

As an example of this standard's flexibility, CIMI does not mandate any authentication nor authorization process, but considering the scope of the EU-SEC framework, there is the possibility to extend the model to include a "*credentials*" resource, in addition of the evidences, to provide access control lists and fine-grained authorization.
This standalone interface shall run behind a reverse proxy (Nginx for example), providing a proper SSL endpoint for the incoming requests and the ability to add authentication on top of CIMI.

Finally, to ensure the auto discovery of resources, there shall exist a well-known entry point resource allowing the discovery of the existing collections and operations (a collection is a group of resources). So in practice, a completely public entry point "*eu-sec-entry-point*" shall be defined and published.

Data transport and privacy

As mentioned previously, the continuous auditing process relies on 3rd party tools performing tests directly on the CSPs' infrastructures, and consequently producing the evidences. On the other end of this process are other 3rd party tools that may act as clients to the evidence storage and retrieve (or even manipulate) the raw data.

The first concern to be addressed is how data will be secured during transport. As it will be described in section 3.3, for security reasons, the evidence storage will have its access points limited to the local machine hosting the database (*localhost*). This protection will be applied though a security group and ensured by the machine's firewall. Given this setup, both evidence producers and consumers will not be able to directly manage data in the database, unless they have been given direct access to the hosting machine.

The recommended way to transport data will therefore be through the CIMI interface described above. Since the interface will be running behind a reverse proxy, on the same host machine as the database, the proxy can be configured to add an extra authentication layer on top of CIMI, which already handles control access to the evidence storage, based on the *credentials* resource.

With CIMI, all operations shall be HTTP based, adding to the usual PUT, GET, DELETE, HEAD and POST requests the possibility to have a JSON body, while covering all basic Search (or Query) and CRUD (Create, Read, Update, and Delete) operations plus the possibility to add custom operations which are mapped into POST requests.

According to the interface standard, the resources' (in this case, evidences) representation shall include an "operations" attribute which explicitly states the actions allowed to the client on that

resource. For EU-SEC this adds an extra layer of robustness as it will allow discrimination of operation based on ACLs – for example, only administrators are able to *EDIT* existing evidences. The use of the universally supported HTTP protocol makes CIMI the right interface for the EU-SEC framework and continuous auditing in general, as it allows easy integration of additional evidence producers and simplifies the evidence management know-how on the consumer(s) side.

3.2 DATA COLLECTION AND STORAGE

The evidence collection and storage will be handled by the CIMI interface described above and an ElasticSearch[3] search engine as the storage element.

ElasticSearch is a document oriented database (written in Java), not following therefore the usual DBMS concepts applied to a relational database. It provides a distributed RESTful search and analytics engine which is not limited by the usual database constraints (schemas, tables, etc.). Raw data is converted into documents with a basic data structure similar to JSON, which makes evidences easier to manipulate. ElasticSearch's Query-DSL (JSON based DSL) can be used to perform fast full text searches, which is very convenient within an automated auditing process as evidences might contain qualitative information. Finally, ElasticSearch architecture is designed so it can be easily scaled horizontally while maintaining an abstraction layer that hides all the distributed storage complexity from the actual end user.

Besides its wide adoption, there's a large community behind ElasticSearch, which comes as an advantage for long term support of the evidence storage.

One can also profit from auxiliary tools, two in particular, which have been created and optimized to be fully integrated with ElasticSearch, providing data processing, transportation and visualization.

Figure 4 below illustrates, on a simplified manner, how the evidence storage architecture will look like. The main path is described by the black arrows while the grey dashed ones represent two other optional workflows.

Besides the ElasticSearch storage element and the CIMI server interface, the evidence storage shall be composed of the following components:

- Logstash[4]: a server-side data processing pipeline that is able to take as input data from multiple sources, simultaneously, filter it and finally send it to one or multiple "storage" elements. There are plenty of official and community provided plugins for Logstash, allowing compatibility with a wide range of data inputs, filtering and data outputs;

[3] https://www.elastic.co/products/elasticsearch
[4] https://www.elastic.co/products/logstash

- Kibana[5]: a data visualization plugin for ElasticSearch, providing plotting and exploring of stored data. It natively supports a wide range of graphical representations;
- Træfik[6]: an HTTP reverse proxy and load balancer that eases the deployment of micro services.

This deployment comprises therefore a full ELK[7] stack (ElasticSearch-Logstash-Kibana), plus a standardized resource management interface and a reverse proxy. The proposed deployment is based on micro services, where each component is completely isolated from all the others.

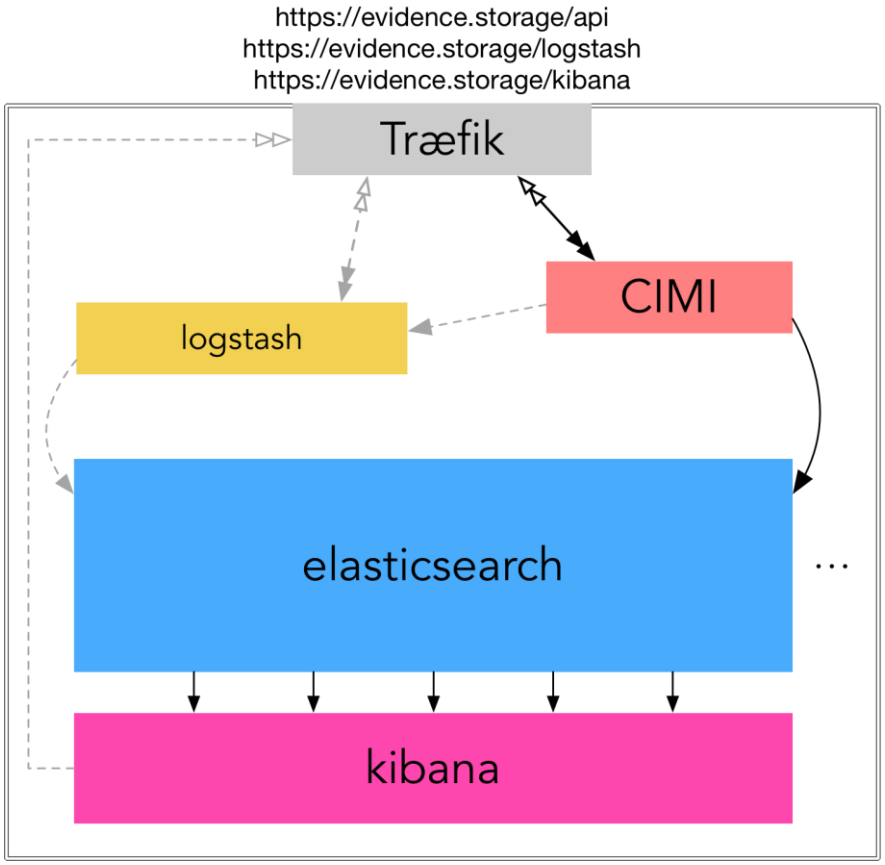

Figure 4 - Evidence storage high level architecture

The proposed workflows for the evidence management are:

1. using solely the CIMI interface to manage all evidences. **This is the main workflow**, represented by the black solid arrows in Figure 4. All CRUD requests are sent to *https://evidence.storage/api*, and are afterwards handled by the CIMI interface which takes care of authenticating and authorizing the request, plus all the request body interpretation and routing to an appropriate action defined within the interface;
2. using CIMI together with Logstash. While in option 1 the CIMI server is responsible for pushing all evidence into ElasticSearch, here the CIMI server would be sending the

[5] https://www.elastic.co/products/kibana
[6] https://traefik.io/
[7] https://www.elastic.co/products

messages through Logstash, which would decrease the interface load and burden of doing consecutive PUT requests into ElasticSearch directly;

3. using solely Logstash. Logstash can also be exposed through Træfik, which means any client could simply offload the raw evidence JSON files directly into data processing pipeline. This option is less desired as it would require some extra fine tuning of the components in order to have proper authentication on top of Logstash.

Note that both options 2 and 3 only consider the injection of messages into ElasticSearch. All the remaining operations (EDIT, DELETE, etc.) would still need to be interfaced through CIMI. Thus, the preference for workflow number 1.

With this architecture, one ensures that direct access to the ElasticSearch storage is never directly allowed from outside the hosting machine, as its endpoint will not be exposed but rather only available internally to the other components.

A final consideration to be made are the mappings to be applied to the ElasticSearch index(es) that will host the evidences and other possible resources. Even though ElasticSearch is able to infer the mapping from the submitted raw data, it is preferable to pre-define a mapping that matches perfectly with the type of data the evidences will carry. Like this, it is possible to assign, beforehand, certain evidences' keywords that will be used as aggregation terms during the data analysis stage, enhancing the querying performance.

Mapping for the Evidences Index in ElasticSearch

This mapping should match the data structure defined in section 2.1.2 and exemplified in Appendix A, while complying with the CIMI standards. The JSON structure proposed in Appendix B provides a base skeleton for this mapping.

3.3 EVIDENCE DISPLAY/PUBLICATION

In IT, collecting data that will never be used or read is the same as carrying dead weight which might on a long-term impact the overall performance of the framework. For this reason, all evidences being stored in ElasticSearch shall be made available for modifications and retrieval. One might identify two different data visibility scopes:

- **Private:** whereby the evidence storage is maintained by a company which uses that data for internal purposes only;
- **Restricted:** whereby the evidence storage is maintained by company or consortium, which have created the appropriate authentication and authorization infrastructure so that other partners and clients can make use of those evidences (an example would be, to create a service catalogue through which end users can optimize the selection of their cloud provider based on the auditing evidences and claims).

By default, the CIMI interface described above shall already provide basic operations for GET, DELETE and EDIT resources, with proper access control.
In order for an authenticated user to search and query evidences from ElasticSearch, CIMI queries shall be used.
One other way to visualize data is to make use of Kibana. By default, this component shall not be publicly exposed, but can be if necessary. This approach would however make all evidences publicly available to anonymous users as authentication is a paid feature of this software.

CIMI-defined Queries

The CIMI specification provides advanced features for manipulating results when searching collections (groups of resources). All the resource selection parameters are specified as HTTP query parameters. These are specified directly within the URL when using the HTTP GET method. For the evidence storage, apart from any other custom actions, the CIMI interface shall provide the possibility to at least:

- filter collections, e.g. "? $*filter=expression*", where "expression" is a mathematical expression compliant with the EBNF grammar defined in the CIMI specification;
- sort collections, e.g. "*$orderby=attributeName[:asc|:desc],...*" ;
- define a range of resources (paging), e.g. "*?$first=number&$last=number*";
- specify a subset of a resource to be acted upon, e.g. "*?$select=attributeName,...*";
- expand references to avoid repeated requests to get referenced resources, e.g. "*?$expand=attributeName,...*".

4 REFERENCES

Cimato, S. a. (2013). Towards the certification of cloud services. *9th IEEE World Congress on Services (SERVICES)*.

Krotsiani, M. a. (2013). Incremental certification of cloud services. *7th International Conference on Emerging Security Information, Systems and Technologies (SECURWARE)*.

Schiffman, J. a. (2013). Cloud verifier: Verifiable auditing service for iaas clouds. *9th IEEE World Congress on Services (SERVICES)*, pp. 239-246.

Stephanow, P. a. (2015). Language Classes for Cloud Service Certification Systems. *11th IEEE World Congress on Services (SERVICES)*.

Stephanow, P. a. (2015). Towards continuous certification of Infrastructure-as-a-Service using low-level metrics. *12th IEEE International Conference on Advanced and Trusted Computing (ATC)*.

Stephanow, P. a. (2017). Evaluating the performance of continuous test-based cloud service certification. *17th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGrid)*, pp. 1117-1126.

Wirth, N. (1996). Extended backus-naur form (EBNF). *ISO/IEC*.

Sample of a test result (and evidence) from a port scan test:


```
{
 "_id": "43d8f64e7251ef3d",
 "className": "de.fraunhofer.aisec.clouditor.testsuite.TestSuiteResult",
 "testId": "c1212655-2f2a-408f-96c8-03736eca52c4",
 "testCaseResults": [
  {
   "details": {
    "accessiblePorts": [
     "22",
     "80",
     "443",
     "8649"
    ]
   },
   "_id": "212987d337a95472",
   "source": {
    "className": "de.fraunhofer.aisec.clouditor.testcases.security.PortScanTestCase",
    "expectedPorts": [
     "22",
     "80",
     "443"
    ],
    "host": "10.244.250.98",
    "runAsRoot": false,
    "timeout": NumberLong(60000),
    "toolName": "nmap",
    "order": 0,
    "service": "LoadBalancer",
    "name": "PortScanTestCase"
   },
   "startTime": new Date(1502880044912),
   "endTime": new Date(1502880045286),
   "passed": true
  }
 ],
 "source": {
  "interval": [
   120,
   240
  ],
  "iteration": -1,
  "label": "normal",
```

```
  "offset": 2,
  "randomized": true,
  "testCases": [
   {
    "className": "de.fraunhofer.aisec.clouditor.testcases.security.PortScanTestCase",
    "expectedPorts": [
     "22",
     "80",
     "443"
    ],
    "host": "10.244.250.98",
    "runAsRoot": false,
    "timeout": NumberLong(60000),
    "toolName": "nmap",
    "order": 0,
    "service": "LoadBalancer",
    "name": "PortScanTestCase"
   }
  ],
  "timeout": 60,
  "name": "TestSuite"
 },
 "startTime": new Date(1502880044912),
 "endTime": new Date(1502880045287),
 "passed": true
}
```


Sample of a test result (and evidence) from a TLS scan test:


```
{
  "_id": "1de41b1a656bf04e",
  "className": "de.fraunhofer.aisec.clouditor.testsuite.TestSuiteResult",
  "testId": "8fbaca32-a2fc-41d6-97b6-dff42761c597",
  "testCaseResults": [
   {
    "className": "de.fraunhofer.aisec.clouditor.testcases.security.TLSScanResult",
    "hasHeartBleed": false,
    "hasTLSFallbackSCSV": false,
    "isVulnerableToOpenSSLCCSInjection": false,
    "hasSecureSessionRenegotiation": true,
    "isVulnerableToCrime": false,
    "trustedCertificate": false,
    "cipherSuites": [
     "DHE-RSA-AES128-SHA256",
     "DHE-RSA-AES256-SHA256"
    ],
    "_id": "a9a2c8517435f06b",
    "source": {
```

```
      "className": "de.fraunhofer.aisec.clouditor.testcases.security.TLSScanTestCase",
      "expectedBlacklist": [
        "TLS_RSA_WITH_DES_CBC_SHA"
      ],
      "sslPort": 443,
      "host": "10.244.250.98",
      "runAsRoot": false,
      "timeout": NumberLong(60000),
      "toolName": "sslyze",
      "order": 0,
      "service": "LoadBalancer",
      "name": "TLS Security Scan"
    },
    "startTime": new Date(1502879926096),
    "endTime": new Date(1502879927335),
    "passed": false
  }
 ],
 "source": {
  "interval": [
    120,
    240
  ],
  "iteration": -1,
  "label": "normal",
  "offset": 2,
  "randomized": true,
  "testCases": [
    {
      "className": "de.fraunhofer.aisec.clouditor.testcases.security.TLSScanTestCase",
      "expectedBlacklist": [
        "TLS_RSA_WITH_DES_CBC_SHA"
      ],
      "sslPort": 443,
      "host": "10.244.250.98",
      "runAsRoot": false,
      "timeout": NumberLong(60000),
      "toolName": "sslyze",
      "order": 0,
      "service": "LoadBalancer",
      "name": "TLS Security Scan"
    }
  ],
  "timeout": 60,
  "name": "testSuite1"
 },
 "startTime": new Date(1502879926095),
 "endTime": new Date(1502879927336),
 "passed": false
```

```
}
```

APPENDIX B

Proposed ElasticSearch index mapping for the test results and evidences:

```
{
  "_index": text,       # eg: evidences
  "_type": text,        # eg: test-result
  "_id": text,          # We can let ES fill it. But ideally, we can set it to TestSuiteResult.testId. This makes it easier to
perform updates
  "_version": double,   # default: 1
  "_score": null,
  "_source": {
    "@timestamp": date,     # eg: 2017-08-15T11:43:01.798Z
    "port": long,           # auto filled if using logstash
    "@version": text,       # auto filled
    "host": keyword,        # auto filled. should match the sinking host
    "message": {
      "id": keyword,          # Can be the same as _id . eg: evidence/0ae59215-b4ce-464e-b84c-50a685da1a4b
      "name": keyword,        # random string to name the test
      "description": keyword, # random text description
      "className": keyword,   # eg: de.fraunhofer.aisec.clouditor.testsuite.TestSuiteResult
      "startTime": date,      # human readable
      "endTime": date,        # human readable
      "passed": boolean,      # true or false
      "updated": date,        # ~=@timestamp, unless the record is updated
      "acl": {
        "owner": {
          "principal": text,    # user of the resource owner. eg: johndoe
          "type": text,         # type of principal. eg: USER/ROLE/BOT
        },
        "rules": [          # a nested ES field
          {
            "principal": text,    # eg: little_johndoe
            "right": text,        # eg: "VIEW"
            "type": text          # eg: "USER"
          },
          {
            "principal": text,    # eg: "Clouditor:is_admin"
            "right": text,        # eg: "ALL"
            "type": text          # eg: "ROLE"
          }
        ]
      },
      "testType": keyword,        # type of test. eg: TLSScanTestCase
      "resourceURI": text,        # eg: "http://clouditor.com/dev/Tests"
      # the next attribute needs further discussion of the generation and storage of claims
```

```
    "claim": {              # a reference to the respective claim
      "href": keyword       # eg: claim/e3db10f4-ad81-4b3e-8c04-4994450da9e3
    },
    "testCaseResults": [    # a nested ES field
      {
        "testCase": keyword,              # eg: "de.fraunhofer.aisec.clouditor.testcases.security.TLSScanResult",
        "hasHeartBleed": boolean,
        "hasTLSFallbackSCSV": boolean,
        "isVulnerableToOpenSSLCCSInjection": boolean,
        "hasSecureSessionRenegotiation": boolean,
        "isVulnerableToCrime": boolean,
        "trustedCertificate": boolean,
        "cipherSuites": text,       # eg: ["DHE-RSA-AES128-SHA256", "DHE-RSA-AES256-SHA256"]
        "details": {
          "accessiblePorts": text,  # can be a list like ["22","80","443"]
        }
      }
    ],
    "source": {
      "interval": long,       # eg: [120, 240]
      "iteration": long,      # eg: -1
      "label": keyword,       # eg: "normal"
      "offset": long,         # eg: 2
      "randomized": boolean,
      "expectedBlacklist": text,  # eg: ["TLS_RSA_WITH_DES_CBC_SHA"]
      "sslPort": long,        # eg: 443
      "expectedPorts": text,  # eg: ["22", "80", "443"]
      "host": ip,             # eg: 10.244.250.98
      "location": geo_point,  # Logstash does this automatically
      "runAsRoot": boolean,
      "timeout": long,        # eg: 60000
      "toolName": keyword,    # eg: "nmap"
      "order": long,          # eg: 0
      "service": keyword,     # eg: "LoadBalancer"
      "name": keyword,        # eg: "PortScanTestCase"
      "timeout": long         # eg: 60
    }
  }
 }
}
```