



City Research Online

City St George's, University of London

Citation: Krotsiani, M., Spanoudakis, G. & Kloukinas, C. (2015). Monitoring-Based Certification of Cloud Service Security. Lecture Notes in Computer Science, 9415, pp. 644-659. doi: 10.1007/978-3-319-26148-5_44

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/13219/>

Link to published version: https://doi.org/10.1007/978-3-319-26148-5_44

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to [City Research Online policy](#).

Monitoring-Based Certification of Cloud Service Security

Maria Krotsiani¹, George Spanoudakis¹, Christos Kloukinas¹

¹Department of Computer Science
City University London
London, United Kingdom
{Maria.Krotsiani.1, G.E.Spanoudakis, C.Kloukinas}@city.ac.uk

Abstract. In this paper, we present a novel approach to cloud service security certification. This approach could be used to: (a) define and execute automatically certification models, which can continuously and incrementally acquire and analyse evidence regarding the provision of services on cloud infrastructures through continuous monitoring; (b) use this evidence to assess whether the provision is compliant with required security properties; and (c) generate and manage digital certificates confirming the compliance of services if the acquired evidence supports this. We also present the results of an initial experimental evaluation of our approach based on the MySQL server and RUBiS benchmark.

1 Introduction

Cloud technology offers a powerful approach to the provision of infrastructure, platform and software services without incurring a considerable cost of owning, operating and maintaining the computational infrastructures required for this purpose. However, despite being cost effective, this technology has raised concerns regarding the security, privacy, governance and compliance of the data and software services offered through it. This is due to the fact that the internals of service provision are not visible to service consumers, and service providers are reluctant to take full responsibility for the security of services that they offer through clouds, and accept liability for security breaches [2]. In such circumstances, there is a trust deficit that needs to be addressed.

The potential of certification as a means of addressing the lack of trust regarding the security of different types of software (and hardware) systems, including the cloud, has been widely recognised [19]. However, the recognition of this potential has not led to as a wide adoption as it was expected originally. The reason for this is that certification has traditionally been carried out through standards and certification schemes (e.g., ISO27001 [19], ISO27002 [19] and Common Criteria [7]), which involve predominantly manual systems security auditing, testing and inspection processes. Such processes tend to be lengthy and have a significant financial cost, which often prevents new and smaller technology vendors from adopting it [11].

The certification of cloud services is not an exception to this overall trend. On the contrary, most of the existing certification schemes (e.g., STAR [27] and OCF [8]) are

not fit-for-purpose for the certification of cloud services. This is due to several reasons. Firstly, current schemes offer no automation and can only support certification at distinct time points without considering the continuum of service provision between these points. Secondly, they produce certificates based on testing without incorporating real and continuous cloud service monitoring. Finally, they cannot support dynamic changes in the structure, deployment and configuration of the systems and data that underpin the provision of cloud services as, for example, the dynamic migration of data and software components across different computational nodes within a cloud infrastructure or a cloud federation.

In this paper, we present a novel approach to cloud service certification. This approach can be used to: (a) define and execute automatically certification models, which can continuously and incrementally acquire and analyse evidence regarding the provision of services on cloud infrastructures through continuous monitoring; (b) use this evidence to assess whether the provision is compliant with required security properties; and (c) generate and manage digital certificates confirming the compliance of services if the acquired evidence supports this. Our approach has been developed as part of the EU R&D project CUMULUS and has been implemented as part of the prototype certification infrastructure of it [10]. An early account of our approach was introduced in [18] and examples of different types of certification models based on it have been presented in [17] and [16]. In this paper, we present an advanced version of our approach, incorporating an elaborated scheme for: (i) assessing the sufficiency of evidence for producing certificates and (ii) executing certification processes according to precisely defined models of them. We also present the results of an initial experimental evaluation of our approach.

The rest of this paper is organized as follows. Section 2 gives an overview of the CUMULUS approach to certification. Section 3, 4 and 5 describe three key ingredients of the certification models that drive the certification process in CUMULUS, i.e., the specification of security properties, the specification of the evidence assessment scheme in such models and the specification of the certification process model, respectively. Section 6 discusses the results of an initial experimental evaluation of our approach. Finally, Section 7 reviews related work and Section 6 summarizes our approach and provides directions for future work.

2 Overview of CUMULUS

CUMULUS has developed an infrastructure supporting the collection and analysis of different types of evidence, including for example test and monitoring data for cloud service provision, as well as data gathered from trusted platform modules [10]. The developed infrastructure can be used by certification authorities to generate and manage digital security certificates for cloud services. It can also be used by cloud service providers operating at different levels of the cloud stack, i.e., cloud infrastructure, platform and/or software service providers for self-certification.

The use of this CUMULUS infrastructure for different types of cloud services and security properties and by different types of cloud service providers is enabled

through the specification of appropriate *certification models*. These models describe the process of collecting and analysing evidence in order to assess security properties and the process of creating and managing digital certificates asserting the outcomes of this process. More specifically, a certification model specifies:

- (i) the cloud service to be certified (i.e., the target of certification (TOC));
- (ii) the security property to be certified for TOC;
- (iii) the certification authority that will sign the certificates generated by the model;
- (iv) an assessment scheme that defines general conditions regarding the evidence that must be collected for being able to issue a certificate;
- (v) additional validity tests regarding the configuration of the cloud provider that must be satisfied prior to issuing certificates;
- (vi) the configurations of the agents that will be used in order to collect the evidence required for generating certificates;
- (vii) the way in which the collected evidence will be aggregated in certificates (evidence aggregation); and
- (viii) a life cycle model that defines the overall process of issuing certificates.

Our monitoring-based approach for certification has been developed as part of the CUMULUS infrastructure and is based on monitoring-based certification models (MBCM) in order to specify and drive the execution of the certification process. Such models incorporate the items (i)-(viii) listed above and are specified in an XML-based language whose top-level structure is shown in Fig 1.

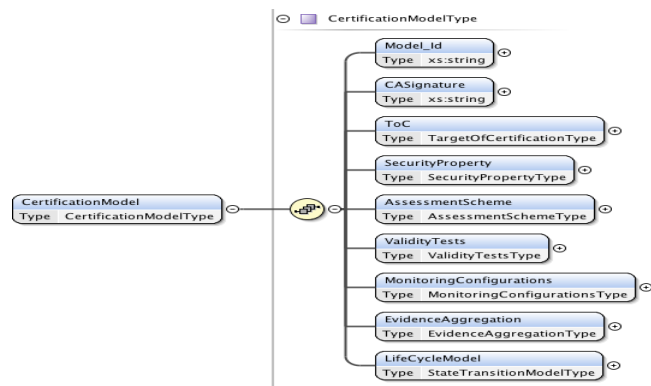


Fig. 1. – Monitoring-based Certification Model schema elements

In the following, we introduce the elements in MBCMs, which are essential for understanding the realization of our approach, namely the specification of security properties, evidence assessment schemes and the process of certification (also known as life cycle model).

3 Specification and monitoring of security properties

In MBCMs, a TOC is specified as a concrete endpoint with a set of service interfaces that are offered by it to external parties (*provided interfaces*), and a set of interfaces required of external parties (*required interfaces*).

The security property to be certified for TOC is specified by one monitoring rule and zero or more assumptions:

$$\text{Security-property} := \text{MonitoringRule} [“,” \text{MonitoringAssumption}]^*$$

In a security property specification, monitoring rules are assertions expressing conditions that must be satisfied during the monitoring of TOC, whilst monitoring assumptions are assertions, which are used to record and update state variables indicating the state of TOC during monitoring. Both monitoring rules and assumptions are expressed as *assertions* in *EC-Assertion+*. *EC-Assertion+* is an extension of *EC-Assertion*, i.e., the language for expressing monitoring conditions in the EVEREST monitoring system [26], which is part of the CUMULUS framework. *EC-Assertion+* is based on Event Calculus [25]. Within it, assertions are formulas of the form:

$$\text{Assertion} ::= [\text{precondition}]^* \text{“}\Rightarrow\text{” } \text{postcondition}$$

The (optional) *precondition* element in an assertion determines the conditions under which the assertion should be checked. The meaning of the *postcondition* element depends on whether the assertion is a monitoring rule or an assumption. In assertions expressing monitoring rules, *postcondition* determines the conditions that are guaranteed to hold (i.e., should be true if the preconditions are true). In assertions expressing monitoring assumptions, *postcondition* determines the states of the system that can be inferred to be true if the preconditions are true.

Both monitoring rules and assumptions are defined in terms of *events* and *fluents*. An event is something that occurs at a specific instance of time and has instantaneous duration. Fluents represent system states and are initiated and terminated by events. The basic predicates used by *EC-Assertion+* are:

- *Happens(e,t,[L,U])* – This predicate denotes that an event *e* of instantaneous duration occurs at some time point *t* within the time range *[L,U]*. An event *e* is specified as *e(id,_snd,_rcv,TP,_sig,_src)* where *id* is its unique id of it, *_snd* is its sender, *_rcv* is its receiver, *_sig* is its signature, and *_src* is the source where *e* was captured from. *TP* denotes the type of the event. *EC-Assertion+* supports three event types: (a) captured operation calls (REQ), (b) captured operation responses (RES) and (c) forced operation execution events (EXC), i.e., operation executions triggered by the monitor itself. EXC events constitute one of the extensions of *EC-Assertion+* over its predecessor *EC-Assertion*. When such events are encountered in a formula, EVEREST attempts to execute the operation defined by *_sig* (by invoking an external service) and, if successful, it replaces any output parameters of the operation with the values produced by it and considers the relevant *Happens* predicate to be true. If the call to the external operation fails, the *Happens* predicate is considered to be false. EXC events are used to execute external computations (e.g., online tests) during monitoring.

- *Initiates(e,f,t)* – This predicate denotes that a fluent *f* is initiated by an event *e* at time *t*. fluents are expressed as n-ary relations of the form *relation(arg₁, ..., arg_n)*, where *arg_i* can be constant values or variables of basic data types.
- *Terminates(e,f,t)* – This predicate denotes that a fluent *f* is terminated by an event *e* at time *t*.
- *HoldsAt(f,t)* – This is a derived predicate denoting that a fluent *f* holds at time *t*. *HoldsAt(f,t)* is true if *f* has been initiated by some event at some time point *t'* before *t* and has not been terminated by any event within *[t',t]*.
- *<rel>(x,y)* – These are relational predicates (*<rel> ::= = | < | > | ≤ | ≥ | ≠*) enabling comparisons between variables of basic data types, or between such variables and constant values.

To demonstrate the use of *EC-Assertion+* in specifying security properties, consider an example showing how it may be used to specify a security property included in the Protection Profile for Database Management Systems developed by Oracle [DBMS PP, 2000] (i.e., a Common Criteria (CC) profile developed for the certification of relational database management systems). This security property (also known as *security functional requirement* or SFR in the context of CC [7]) is about the timing of user identification and is expressed as follows within the protection profile:

FIA_UID.1.2: *The TSF shall require each DATABASE user to be successfully identified before allowing any other TSF-mediated actions on behalf of that DATABASE user.*

The certification model for monitoring and certifying FIA_UID.1.2 consists of three assertions: two assumptions and one monitoring rule. The two assumptions in the MBCM are used to initialise and terminate a fluent indicating whether a user is connected to the DBMS following successful authentication. The fluent is expressed by the relation *Connected(_thread-id, _user)*. The meaning of the relation is that the user indicated by the variable *_user* has been connected to the DBMS through the thread indicated by the variable *_thread-id*. The fluent *Connected(.)* is initiated when an event showing the successful connection of *_user* to the DBMS occurs. The assumption that is used to initiate the fluent is expressed as¹:

FIA_UID.1.2.A1

```
Happens(e(_eId,_thread-id,_host,REQ,o(_thread-id,_query-id,
_queryType,_user),_SRC),_t1,R(_t1,_t1)) ∧ (_queryType = Connect) ⇒
Initiates(e(_eId, _thread-id, _host, REQ, o(_thread-id,_query-id,
_queryType,_user),_SRC), Connected(_thread-id, _user),_t1)
```

The above assertion monitors events of the form *o(_thread-id, _query-id, _queryType, _user)*. When an event of this form occurs during the operation of the DBMS and the type of the query captured by the event (i.e., *_queryType*) is “Connect”, the state *Connected(.)* is initiated. The events *o(_thread-id, _query-id, _queryType, _user)* required in order to operate the certification model of this example are captured during the operation of the DBMS to be certified and are passed to

¹ For readability, we provide the specification of assertions in the high level syntax of *EC-Assertion+*. *EC-Assertion+* has also an XML schema used in actual monitoring.

the CUMULUS framework by an event translator that we have developed for this purpose (see Sect 4). The state *Connected(.)* may also be terminated during the operation of a DBMS if a given user disconnects from the DBMS. The assumption that captures such disconnection events and updates the fluent *Connected(.)* is expressed as:

FIA_UID.1.2.A2

```
Happens(e(_eId,_thread-id,_host,REQ,o(_thread-id,_query-id,
_queryType,_user),_SRC),_t1,R(_t1,_t1)) ^ (_queryType = Quit) ^
HoldsAt(Connected(_thread-id,_user),_t1) ⇒
Terminates(e(_eId,_thread-id,_host,REQ,o(_thread-id,_query-id,
_queryType,_user),_SRC),Connected("thread-id","user"),_t1)
```

According to above assertion, the fluent *Connected(.)* is terminated, when a “Quit” event occurs for a user, provided that at the time when the “Quit” event the particular user is connected. This is checked in the formula by the *HoldsAt(Connected(_thread-id,_user),_t1)* condition.

The monitoring rule assertion that is used to check if a DBMS satisfies FIA_UID.1.2 is expressed as:

FIA_UID.1.2.MR1

```
Happens(e(_eId,_thread-id,_host,REQ,o(_thread-id,_query-id,
_queryType,_user),_SRC),_t1,R(_t1,_t1)) ^
not (_queryType = Connect) ⇒
HoldsAt(Connected(_thread-id,_user),_t1)
```

The above rule monitors if at each time (*t1*) when a user executes queries at the DB server, which are not of type “*Connect*”, he/she must have been successfully connected to the server. Thus, the monitoring rule checks that when queries of a type other than “*Connect*” occur, the fluent *Connected(_thread-id,_user)*, which indicates that the user has already established a connection to the server through the specific thread, holds.

3.1 Specification and verification of assessment scheme

The assessment scheme in a MBCM defines conditions regarding the sufficiency of evidence that must be collected in order to be able to issue a certificate. These conditions are related to: (i) the sufficiency of the extent of the collected evidence, and (ii) anomalies and conflicts that should be monitored during the certification process. In this paper we do not discuss anomalies and conflicts but an account of them is available in [17].

Evidence sufficiency conditions may be specified as: (a) the minimum period of monitoring TOC, (b) the minimum number of monitoring events, and/or (c) the representativeness of the monitoring events with respect to the expected behaviour of TOC that should be seen by the monitor before a certificate can be issued. Whilst the specification of (a) and (b) is straightforward, to enable checks of the representativeness of monitoring events, the certification model should include a specification of a model of

the *expected behaviour* of TOC (i.e., an ETOCB model). This model is specified as a deterministic automaton with expected relative event frequencies of the form:

$$\text{ETOCB} = \langle \text{States}, \text{Events}, s_{\text{init}}, \text{PTrans}, \text{FinalStates} \rangle$$

In the ETOCB specification: *States* is the (finite) set of TOC states that are critical for the monitoring process; *Events* is the set of all possible events the TOC may produce that are of interest to certification; s_{init} is the initial TOC state; *PTrans* is a finite set of labelled transitions between two states; and *FinalStates* is the set of states where the certification automaton terminates. *PTrans* includes elements of the form $(os, ds, e, R(lpr, upr))$ where *os* is the origin state of the transition, *ds* is the destination state of the transition, *e* is the signature of the event triggering the transition, and $R(lpr, upr)$ is the range of the expected relative frequency of undertaking this transition whilst the system is in *os* ($R(lpr, upr)$ can be: (lpr, upr) , $[lpr, upr)$, $(lpr, upr]$ or $[lpr, upr]^2$). The ETOCB model must satisfy some constraints. In particular: (i) *e* must be an element of *Events*, i.e., an event denoting the invocation (or the response produced following an invocation) of an operation in the *provided* interface of TOC; (ii) the boundaries *lpr*, *upr* should satisfy the conditions: $0 \leq lpr, upr \leq 1$, and $lpr \leq upr$; and (iii) ETOCB must be a deterministic model.

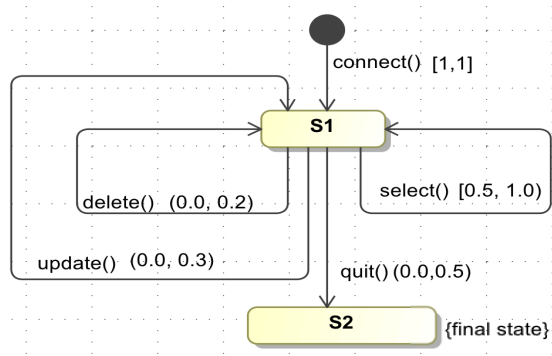


Fig. 2. Expected System Operation Model for Monitoring-Based Certification Model

ETOCB defines events that should be seen at different states during the operation of TOC (i.e., executed operations of the ToC) for the monitoring evidence to be sufficient. For certifying MySQL server, for example, this evidence should include executions of select, update, delete, and quit MySQL commands with specific frequencies. ETOCB is not required to be a complete model of TOC's behaviour; it only needs to define the states and events of importance for the property to be certified.

Fig. 2 gives an example of the ETOCB model for a relational DB server. This model expresses a view about the typical range of the server usage that should be taken into account in the certification of the server. According to it:

² “[” and “]” denote a closed range at the lower and upper boundary respectively, and “(” and “)” denote an open range at the lower and upper boundary respectively.

- The first interaction with the TOC should be a *connect* call to it (see event of transition from *InitialState* to *S1*) since a connection to the server should be established before any other query occurs. Also, according to the frequency range of this transition (i.e., [1,1]), *connect* calls should be the only initial event in any monitoring event trace, for the trace to be considered valid for the purposes of certification.
- Once a connection to the server is established, interactions with it may be requests for the execution of *select()*, *update()*, *delete()* or *quit()* operations (i.e., SQL queries) with expected frequency ranges [0.5, 1.0), (0.0, 0.3), (0.0, 0.2), and (0.0, 0.5), respectively, as indicated by the relevant transitions from *S1* to *S1* and *S2*. These expected frequency ranges require that data retrieval events (*select()* queries) will constitute at least half of the interactions with the server but data *update()* and *delete()* queries should also be seen. The model also expresses that: (a) it will be sufficient for certification purposes to see an event trace with update queries up to below 30% and delete queries up to below 20% of all interactions, and (b) whilst at *S1*, the user may decide to *quit()* (see transition from *S1* to *S2*). Also, the *lpr* of the latter transition (i.e., $lpr > 0$) reflects that an event trace must always end with a *quit()* request for it to be a valid event trace for certification.

Table 1. Algorithm for checking compliance of event traces with ETOCB

```

CheckEvent(e, state, nstate, CountES[e,state], Counts[state], valid) {
  // CountES[e,state] is the total number of occurrences of e in state
  // Counts[state] is the total number of occurrences of any event in state
  if there is t in state.transitions such that t.event = e then {
    CountES[e,state] = CountES[e,state] + 1;
    Counts[state] = Counts[state] + 1; nstate = t.ds; valid = true }
  else
    {valid = false}
  }
Boolean UpdateCounts(trace){
  /* ValidPR[e,s] indicates the satisfaction of expected frequency range of
  all events of all state transitions */
  Set CountES[e,s] to 0 for all states s and events e of its transitions;
  Set Counts[s] to 0 for all states s;
  Set ValidPR[e,s] to false for all states s and events e of its transitions;
  CST = ETOCB.s0; //CST is the current state
  NST = nil;
  validTrace = true;
  while not end of event trace and validTrace do {
    e = next non processed event in trace;
    CheckEvent(e, CST, NST, CountES[e,CST], Counts[CST], validTrace);
    if validTrace {
      for each t in CST.transitions do {
        if (CountES[t.e,CST]/Counts[CST] in R(t.e.lpr, t.e.upr)) {
          ValidPR[t.e,CST] = true}
        }
      CST = NST
    }
  }
  return (validTrace)
}

```

The existence of the ETOCB model enables the CUMULUS infrastructure to check if a representative sample of the behaviour of TOC has been considered before

a certificate can be issued. The check of the coverage of ETOCB by the stream of TOC events that has been processed by CUMULUS is carried according to the algorithm of Table 1. The algorithm *UpdateCounts()* in the figure checks if each next event in the event trace is consistent with the ordering of events in ETOCB. If it is not, *UpdateCounts()* reports the trace as invalid (as relative frequencies would not matter). If an encountered event is valid, *UpdateCounts()* updates the relative frequency of it in the current state (see array *CountES[e,state]*). It also updates the array *ValidPR[e,s]*, which indicates if the expected frequency range of the current event *e* in state *s* is preserved by the current relative frequencies of events. *ValidPR[e,s]* can be checked once all other sufficiency conditions (e.g., period of monitoring) are established to check if the coverage sufficiency conditions w.r.t. ETOCB are also satisfied.

3.2 Specification and execution of life cycle model

The life cycle model (LCM) in a certification model defines the process by which certificates can be generated and managed (e.g., monitored, issued, suspended, revoked). LCM is a compulsory element of a certification model as it enables a certification authority to specify with full precision the certification process, by defining the different states of certificates that can be generated by the certification model and which events should change it. During the operation of the CUMULUS framework, the LCM is used to monitor on-going certification processes, determine the state at which they are (e.g., collecting monitoring evidence, checking validity conditions prior to issuing a certificate) and, depending on it, update the state of the certificate that may be generated by the process.

A life cycle model (LCM) is defined as a state transition model of the form

$$\text{LCM} = \langle s_{\text{init}}, \text{States}, \text{Trans} \rangle$$

In an LCM, (i) *States* is the finite set of states of it (a state may be an atomic state or a composite state specified by another embedded LCM); (ii) s_{init} is the initial state of the process; and (iii) *Trans* is a finite set of transitions between two states. *Trans* includes elements of the form (s_i, s_j, e, g, a) where s_i is the origin state of the transition; s_j is the destination state of the transition; *e* is the signature of the event triggering the transition; *g* is guard condition that must be satisfied for the transition to take place; and *a* is a set of actions that should be executed if the transition takes place. In an LCM, *e* must be an element of the *provided* interface of CUMULUS (e.g., the operation enabling the notification of monitoring events, the operation to be executed if the user of the framework wishes to suspend or revoke a certificate).

An example of an LCM is shown in Fig. 3. The LCM in the figure has an initial state called *Activated* and the states *InsufficientEvidence*, *Pre-Issued*, *Issued*, and *Revoked*. It also has two composite states: *Continuous Monitoring* and *Issuing*.

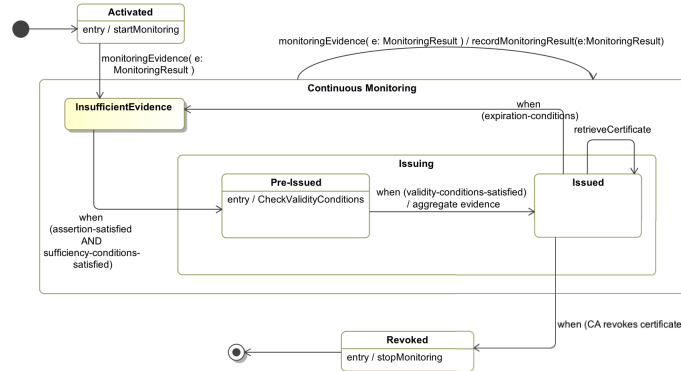


Fig. 3. UML diagram of Life-Cycle Model

According to the model, after a certificate is activated, it moves to the *InsufficientEvidence* state, at which the monitoring evidence that is relevant to it starts getting accumulated. When the accumulated evidence becomes sufficient according to the *EvidenceSufficiencyConditions* specified in the MBCM, and there have been no violations of the monitoring rule that defines the security property (i.e., the security property of the MBCM is satisfied), the certificate moves to the state *Pre-Issued*. At this state, the certification infrastructure will check if the extra validity conditions for the certificate type (if any) are satisfied and, if they are, the certificate will move to the state *Issued*. In this state, any interested party with appropriate authority can retrieve the issued certificate from the CUMULUS infrastructure. Whilst a certificate is at the *Issuing* state, monitoring continues and if a violation of the monitoring rule of the MBCM is detected, the certificate moves to the *Revoked* state at which it will no longer be valid and available. It should be noted, that for readability purposes, in Fig. 3, we have used condition labels that indicate the meaning of the relevant conditions. In the actual specification of LCM, however, conditions are declared by their unique XML level IDs, which enable condition elements to be retrieved and checked against the evidence database of the CUMULUS infrastructure.

The LCM of a certification model is used by the CUMULUS framework to monitor the overall certification process and update the status of certificates that may be generated according to it. More specifically, starting from the initial state of the LCM the framework will process all events according to the model. This processing is based on the algorithm of Table 2. The events received/generated by the framework during the certification process are placed in a queue. An event can be a condition that is met (e.g., *EvidenceSufficiencyCondition*, aggregation period, expiration condition etc.). The algorithm checks if there is an event in the queue that matches an event of a listed transition of the current state of the LCM and if the guard condition of it (if any) is satisfied. When these conditions are satisfied for the specific transition, the algorithm executes the actions for the transition, and sets the status of the certificate that is being handled by the process, to the state that the transition leads to. To check the conditions associated with the transitions of an LCM, the algorithm pulls regularly

data from the database storing the monitoring evidence gathered, and checks the conditions against it (e.g., see condition *assertion-satisfied* in the LCM of Fig. 3).

Table 2. Algorithm of the Life Cycle Manager component

```

State ChooseTransition(State curstate, EventQueue queue){
  top = queue.head(); //returns null when queue is empty
  trev = {t ∈ transitions(curstate): top≠null && t.event()==top};
  //trans matching events
  trem = {t ∈ transitions(curstate): t.event() = ""}; //trans with no events
  enev = {t ∈ trev: satisfied(guard(t))}; //trans with True guard & match event
  enem = {t ∈ trem: satisfied(guard(t))}; //trans with True guard but no event
  t = null;
  if (enev = 0 && enem = 0){
    if (top≠null) throw invalidEvent; //non matching event from the queue
    else return(curstate);
  } else if (enev ≠ 0) { //select transition with event
    t = select (enev);
    queue.pop();
  } else { //select transition with True guard but no event
    t = select (enem);
  }
  for (a : retrieveActions(t)) { //retrieve transition actions
    execute(a); //execute actions
  }
  return t.nextState(); //return the new state
}

```

4 Evaluation

In order to evaluate the performance of our monitoring-based certification approach, we have conducted an experiment based on a case study involving the certification of a real system. The system that we selected was the open source MySQL server [21]. Our choice was influenced by: (a) the complexity of this system, (b) the existence of a Protection Profile generated by Oracle that specifies security properties for such systems based on Common Criteria [9] (aka *Security Functional Requirements (SFR)*), and (c) the existence of benchmarks for creating realistic workloads for the MySQL server that would enable us evaluate our automated certification process in realistic conditions. Moreover, since our approach does not support interventions with the purpose of addressing or restoring security violations, we focus only on the evaluation criteria of the MySQL server, based on the selected Protection Profile.

The experiment that we set up to evaluate our approach realised a certification process for the security functional requirement FIA_UID.1.2 for the MySQL Server, based on a certification model including the assertions as described in Sect. 3. We also used the RUBiS benchmark [24] to produce realistic workloads of events for the MySQL server and monitor the server for certification purposes during the execution of these workloads. RUBiS is an auction site prototype, similar to eBay, which implements the core functionality of an auction site, i.e., selling, browsing and bidding. To capture events (i.e., logs of queries) from the operation of the server, we used the

MySQL AUDIT Plugin developed by McAfee [20]. This plugin captured the logs created during the execution of the RUBiS workloads against the server. The events logged by the plugin were initially exported as *.json* files and subsequently parsed and converted into events, in the *.xml* format required by the CUMULUS infrastructure. All the different systems used in our experiment, including RUBiS, MySQL, EVEREST and CUMULUS were deployed on a cloud cluster involving a test-bed Cloud cluster equipped with four 4-core server machines each running at 2.20GHz, with 8GM of main memory, 450GB of disk space under Ubuntu 3.8.0.

The basic measure that we used in order to evaluate the performance of the certification process was the average time for making a decision about the monitoring assertion formulas in the model, called decision delay or *d-delay*. *d-delay* measures the difference between the time point when the latest event that is needed in order to make a decision about the satisfaction or otherwise of a monitoring formula occurs (t_c) and the time when following the capture and processing of the event, the monitor makes a decision on whether the formula is satisfied (t_p), i.e., $d = t_p - t_c$. Based on *d-delay* measures for individual instances of monitoring formulas, we calculated the average delay in the monitoring process using following formula $ave(d) = \sum d/N$ where: (i) *d* is the *d-delay* of each monitoring rule instance, and (ii) *N* is the total number of monitoring rule instances for which a decision was made.

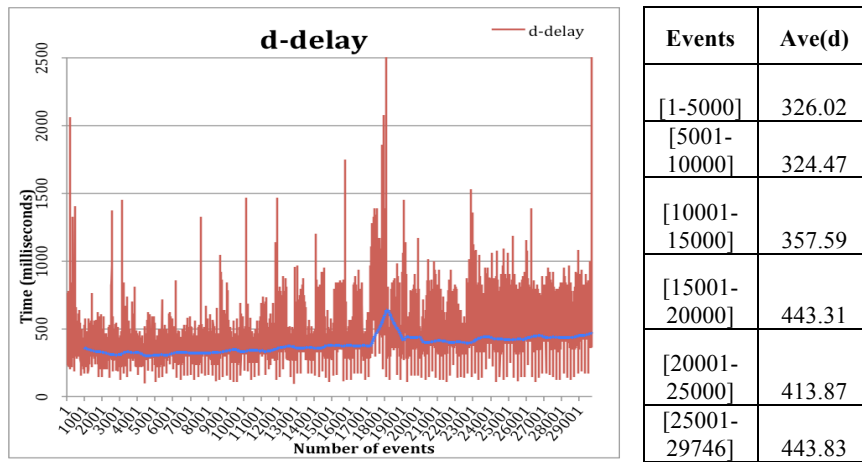


Fig. 4. – *d-delay* in execution of the database certification model

The graph in Fig. 4 shows the *d* values for the different events of the RUBiS benchmark that caused monitoring rule checks in the certification model, and the moving average of *d-delay* ($Ave(d)$) calculated over a window of 1000 events. The average value of *d-delay* across the whole RUBiS benchmark was 384.33 milliseconds (standard deviation = 118.92 milliseconds). As shown in the figure, $ave(d)$ remained relatively stable throughout the execution of the benchmark, showing that certification results can be produced quickly following the actual events.

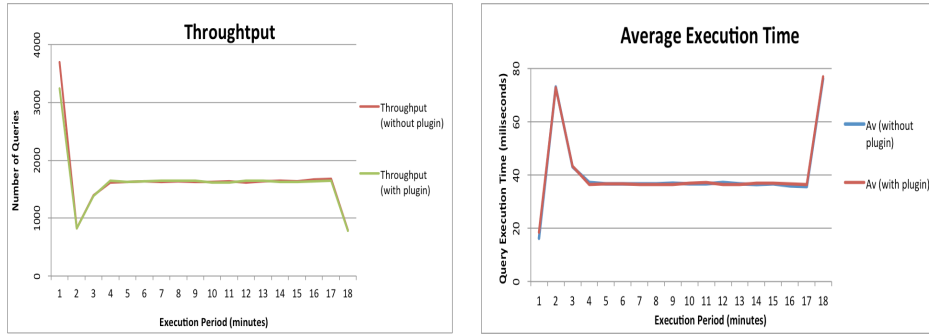


Fig. 5. – Average throughput (i) and query processing time (ii) in executing the RUBIS benchmark on MySQL server with and without the MySQL AUDIT plugin

Table 2. Average throughput and query execution time with and without the AUDIT plugin

Min	Throughput		Average Query Processing Time (msecs)	
	No Plugin	With plugin	No plugin	With plugin
1	3688.9	3245	16.27	18.49
2	819.9	824.7	73.18	72.75
3	1390.9	1386.1	43.14	43.29
4	1615	1651.1	37.15	36.34
5	1630.4	1629.7	36.8	36.82
6	1638.2	1636.5	36.63	36.66
7	1630.7	1645.9	36.79	36.45
8	1633.9	1643.9	36.72	36.5
9	1624.4	1648.5	36.94	36.4
10	1630.2	1619.9	36.81	37.04
11	1630.9	1611.8	36.79	37.23
12	1617	1646.9	37.11	36.43
13	1638.8	1648.5	36.61	36.4
14	1651.5	1627.3	36.33	36.87
15	1631.3	1628	36.78	36.86
16	1665.5	1635.4	36.03	36.69
17	1677.4	1649.8	35.77	36.37
18	788.2	779.6	76.12	76.96

In addition to the time needed to generate certification results, the execution of a CUMULUS monitoring-based certification model may have an impact on the operation of TOC as it is necessary to instrument and/or configure the TOC in order to produce the events needed for the monitoring process that underpins certification. To evaluate this overhead in the case of the MySQL server, we executed the RUBIS benchmark without using the MySQL audit plugin in the server (*case (a)*) and with the use of the MySQL audit plugin in the server (*case (b)*). The overhead was estimated by calculating the average throughput (i.e., the number of queries executed per

minute) of the server in 10 different executions of case (a) and 10 different executions of case (b). Each of these 20 executions involved the execution of the same number of RUBIS queries against the server (~30,000 queries) but the queries executed in each execution were selected randomly by the RUBIS system. The completion of the execution of the different query sets took on average 18 minutes.

The average throughput for cases (a) and (b) was measured per minute and the result is shown in the *Throughput* graph of Fig. 5. As shown in this graph the use of the MySQL AUDIT plugin had almost a very minor effect on the performance of the server. The same is evident from *Average Execution Time* graph in Fig. 5, which shows the average execution time per RUBIS query (in milliseconds), for every minute during the execution period. The absence of any significant effect is also evident from which shows the actual throughput and average query execution times for (a) and (b). The main difference in query execution time was observed only in the initial stage of the execution of each query set, when RUBIS sent queries to establish the connection to MySQL for each transaction thread.

5 Related Work

Research related to our approach includes work for service certification, cloud security and cloud monitoring. In this section we give an overview of this work.

Similar approaches in the field of security certification schemes focus mostly on concrete software components and provide self-assessed, human-readable certificates. As a result, these approaches cannot be integrated into dynamic service processes that require machine-readable certificates. Significant work on the representation and use of digital certificates in SOA systems was done in the FP7 Project ASSERT4SOA. This project developed a test-based certification of software services and a framework for representing and using machine-readable certificates, known as ASSERTS.

Research on the certification of cloud services is still in an early stage. The work of Grobauer et al. [12], assess some vulnerabilities of cloud computing, and outlines the main reason of the existence of such vulnerabilities as the lack of certification schemes and security metrics. Heiser and Nicolett [13] have evaluated the cloud security risks and proposed an IT risks sharing scheme. Furthermore, Anisetti et al [1] presented a trusted model for certifying cloud services, by delegating different dynamic testing mechanisms.

A commonly used framework for cloud certification is CSA's Cloud Controls Matrix (CCM) [5]. CCM contains a comprehensive set of baseline controls to assess the information security assurance level of cloud providers and maps these controls to existing frameworks such as ISO/IEC 27001-2013, PCI DSS Cloud Guideline [23], COBIT [6], NIST [15], or IT Baseline Protection Catalogues [16].

The Cloud Security Alliance has also developed and launched in 2011 the CSA Security, Trust and Assurance Registry (STAR) Program [27], which is a third party independent assessment of the security of a cloud service provider. STAR is based on a multi-layered structure defined by Open Certification Framework (OCF) Working Group [8] and on the requirements of the ISO/IEC 27001 management system stand-

ard together with the CSA CCM. STAR approach consists of three different levels of certification. Our approach is similar to the third level of STAR, which is the CSA STAR Continuous Monitoring, which is meant to enable automation of auditing, assessment, monitoring and certification of security practices of cloud providers.

Cloud monitoring has been supported by several monitoring systems. Most of them, however, focus on monitoring performance and SLAs monitoring rather than security properties (e.g., [15][3]) and do not support security certification.

6 Conclusion

In this paper, we have presented an automated certification approach for cloud services based on continuous monitoring. We have described the core mechanisms of our approach that can be used to specify and realise a certification process for the security of cloud services. We have also given an example showing how the approach can be used to realise, in an automated manner, the certification of a security property defined in a Common Criteria protection profile for database systems.

The certification model underpinning this example has been used to evaluate our approach as part of an experiment in which we used the MySQL server. The results of this evaluation showed that the certification process that we proposed can produce results in an automated manner, fast and without interfering significantly with the performance of the system that is certified. The average time complexity of the monitoring algorithm is $N \cdot M$, where N is the average number of events and M is the average number of rule instances, at different time periods during the monitoring process. M depends on the number of different events in assertions and the time constraints between them. The average delay in checking assertions is experimentally shown not to be significant. Basic security properties (integrity, availability, confidentiality) can be expressed by assertions of such complexity as shown in the literature. Hence, our approach is feasible for large numbers of events. A video of a demo of the implementation of our approach is available from: http://youtu.be/HWb_dA2UCxM.

Our on-going work focuses on a further evaluation of our approach for different types security properties and cloud services. We are also investigating the use of model checking techniques to verify statically properties of certification models.

Acknowledgment

The work presented in this paper has been partially funded by the EU FP7 project CUMULUS (grant no 318580).

References

1. Anisetti, M., Ardagna, C. A. and Damiani, E., "A Certification-Based Trust Model for Autonomous Cloud Computing Systems", Int. Conf. on Cloud and Autonomic Computing (CAC 2014, London, UK, 2014

2. Ardagna C.A., Asal R., Damiani E. and Vu Q. V. "From Security to Assurance in the Cloud: A Survey", *ACM Computing Surveys (CSUR)*, vol. 48, no. 1, Article 2 (July 2015).
3. Barham, Paul, et al. "Xen and the art of virtualization." *ACM SIGOPS Operating Systems Review*. Vol. 37. No. 5. ACM, 2003.
4. Bezzi, M., Sabetta, A., & Spanoudakis, G. "An architecture for certification-aware service discovery". *1st Int. IEEE Workshop on Securing Services on the Cloud*, pp. 14-21, 2011.
5. Cloud Security Alliance, Cloud Controls Matrix, Available from: <https://cloudsecurityalliance.org/research/ccm/>
6. COBIT, IT Assurance Guide: Using COBIT, Control Objectives for Information and related Technology, 2007, information Systems Audit and Control Association.
7. Common Criteria (CC) for Information Technology Security Evaluation, CCDB USB Working Group, 2012, part 1-3. Available from: <http://www.commoncriteriaportal.org>.
8. CSA, "Open Certification Framework". <https://cloudsecurityalliance.org/research/ocf/>
9. Database Management System Protection Profile, Issue 2.1, May 2000, Available from <http://www.commoncriteriaportal.org/files/ppfiles/T129%20-%20PP%20v2.1%20%28dbms.pp%5B1%5D%29.pdf>
10. Egea M., Mahbub K., Spanoudakis G., and Vieira M.R., A Certification Framework for Cloud Security Properties: the Monitoring Path, *Accountability and Security in the Cloud, LNCS 8937*, pp 63-77, DOI: 10.1007/978-3-319-17199-9_3, 2015
11. ENISA, Security Certification Practice in the EU: Information Security Management Systems – A Case Study, v1, October 2013, available from: <https://www.enisa.europa.eu/>
12. Grobauer, B., Walloschek, T. and Stocker, E., "Understanding Cloud Computing Vulnerabilities," *Security & Privacy, IEEE*, vol.9, no.2, pp.50-57, March-April 2011
13. Heiser J. and Nicolett, M., "Assessing the Security Risks of Cloud Computing", Gartner TR, 2008
14. Heiser, J. and Nicolett, M., "Assessing the security risks of cloud computing", p. 1-6, 2008
15. IT Baseline Protection Catalogs, Available from: <http://www.bsi.de/gshb/index.htm>
16. Katopodis S., Spanoudakis G., and Mahbub K. "Towards hybrid cloud service certification models." In *2014 IEEE International Conference on Services Computing*, pp. 394-399
17. Krotsiani, M. and Spanoudakis, G., "Continuous Certification of Non-Repudiation in Cloud Storage Services", 4th IEEE Int. Symp. on Trust and Security in Cloud Computing, 2014
18. Krotsiani, M., Spanoudakis, G. and Mahbub, K., "Incremental Certification of Cloud Services", 7th Int. Conf. on Emerging Security Information, Systems and Technologies, 2013
19. Lagazio M., Barnard-Wills D., Rodrigues R., Wright D., "Certification Schemes for Cloud Computing", EU Commission Report, ISBN 978-92-79-39392-1, DOI:10.2759/64404
20. McAfee MySQL AUDIT Plugin, Available From: <https://github.com/mcafee/mysql-audit>
21. MySQL server, available from: <http://www.mysql.com/>
22. National Institute of Standards and Technology, "Information Security Handbook: A Guide for Managers," NIST Special Publication 800-100, October 2006.
23. Payment Card Industry Data Security Standard (PCI DSS), Available from: https://www.pcisecuritystandards.org/security_standards/documents.php?document=dss_cloud_computing_guidelines
24. RUBiS Benchmark, Available Form: <http://rubis.ow2.org/>
25. Shanahan, Murray. "The event calculus explained." In *Artificial intelligence today*, pp. 409-430. Springer Berlin Heidelberg, 1999.
26. Spanoudakis, G., Kloukinas C., and Mahbub K., "The serenity runtime monitoring framework." In *Security and Dependability for Ambient Intelligence*, pp. 213-237. Springer, 2009
27. STAR Certification, Cloud Security Alliance. Available from: <https://cloudsecurityalliance.org/star/>
28. Vincent C. Emeakaroha, et al. "DeSVi: An Architecture for Detecting SLA Violations in Cloud Computing Infrastructures", 2nd Int. ICST Conference on Cloud Computing, 2010