Assessing the genuineness of events in runtime monitoring of cyber systems

Theocharis Tsigkritis[a]
[a]*CrypteiaNetworks, Athens, Greece*
Email: Theocttsigkritis@crypteianetworks.com

George Spanoudakis[b]
[b]*School of Informatics, City University London, London, United Kingdom*
Email: g.e.spanoudakis@city.ac.uk

***Abstract* —** Monitoring security properties of cyber systems at runtime is necessary if the preservation of such properties cannot be guaranteed by formal analysis of their specification. It is also necessary if the runtime interactions between their components that are distributed over different types of local and wide area networks cannot be fully analysed before putting the systems in operation. The effectiveness of runtime monitoring depends on the trustworthiness of the runtime system events, which are analysed by the monitor. In this paper, we describe an approach for assessing the trustworthiness of such events. Our approach is based on the generation of possible explanations of runtime events based on a diagnostic model of the system under surveillance using abductive reasoning, and the confirmation of the validity of such explanations and the runtime events using belief based reasoning. The assessment process that we have developed based on this approach has been implemented as part of the EVEREST runtime monitoring framework and has been evaluated in a series of simulations that are discussed in the paper.

Keywords — cyber system monitoring, event trustworthiness, belief based reasoning, abductive reasoning

I. INTRODUCTION

Monitoring the preservation of security properties of distributed software systems at runtime is a form of operational verification that complements static or dynamic model checking and system testing. Monitoring is particularly important for security properties in cases where the preservation of the specification of a system by its implementation cannot be guaranteed, or there is a possibility of having runtime interactions between system components, which are difficult to foresee and detect during system design and testing and can create security vulnerabilities. Such circumstances may arise when systems deploy software components outside the ownership and control of the system provider dynamically.

Cyber systems – i.e., systems based on components distributed and deployed over the Internet (and possibly a mix of local area and ad hoc networks) and offering services accessible over it – fall under this category due to the inevitably loose control and distributed ownership of the components that they deploy. Successful security attacks over cyber systems (e.g., denial-of-service attacks, man-in-the-middle attacks), indicate that regardless of how carefully the security of system component interactions and communications has been analysed, it will always be possible to identify system vulnerabilities and lunch successful cyber attacks [51].

Over the last decade, several approaches have been developed to support runtime system monitoring (aka runtime verification). Some of these focus on monitoring security properties (e.g. [18][21][23][26][29][46]) whilst others are aimed at monitoring other general types of system properties (e.g. [5][6][10]). Also when it comes to cyber systems, monitoring tends to focus on the network (as opposed to the application) level [44][45][44][47][48][49].

A limitation of existing approaches arises from the fact that they check properties based on events that they receive at runtime from the system under surveillance without assessing whether these events are genuine and trustworthy. Monitoring can be affected by *faulty events* produced by malfunctioning system components, or malfunctioning captors and transmitters that have responsibility for intercepting and sending them to the monitor. Also, event captors and transmitters as well as the communication lines between them and the monitor might become the target of attacks aiming to produce fake events (*attack events)* and, through them, compromise the monitoring process and, consequently, system security. When any of these compromises occurs and faulty or attack events are sent to monitors, the latter can generate erroneous results.

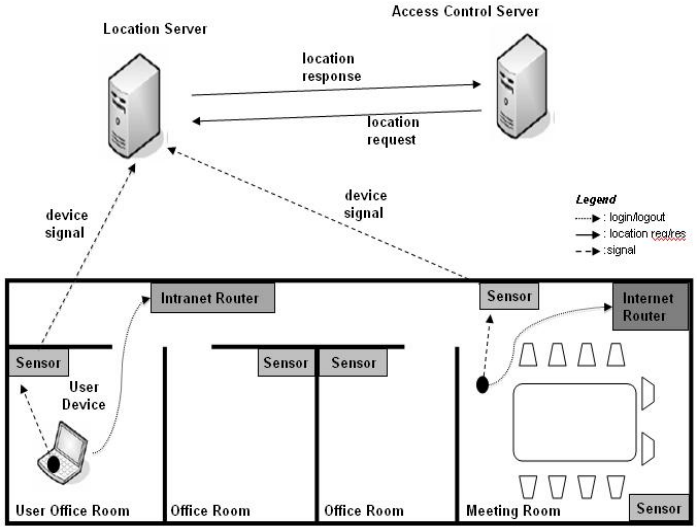

Figure 1. Location based access control system (LBACS)

To illustrate why, consider a system providing location based access control to different resources of an organisation with spatially distributed premises, based on user and device authentication, as well as device location information; this system is an industrial case study introduced in [3] that we will refer to as "Location Based Access Control System" or simply "LBACS" in the following. In LBACS, users who move within the distributed physical spaces of the organisation using mobile computing devices (e.g., PDAs, smart phones) may be given access to different resources (e.g., intranet, printers, scanners), depending on their credentials, the credentials of their devices, and their exact location within the physical space(s). Such resource accessing scenarios arise frequent in systems supporting digital commercial and financial transactions (e.g., systems with mobile points of sale, mobile banking etc.) as well as enterprise systems allowing and/or based on *bring your own device (BYOD)* scenarios [50].

Fig.1 shows the components and physical configuration of LBACS. As shown in the figure, the operation of LBACS is based on several autonomous and distributed components. These include a *location server,* an *access control server*, *sensors* and *Internet routers*. The access control server polls the location server at regular intervals, to obtain the position of the devices of all the users who are logged on to LBACS. LBACS compliant devices can log on to it through the wireless network available in the physical space controlled by LBACS (not the sensors) and must have installed daemons sending signals to the location server via the location sensors, periodically. Based on these signals, the location server can calculate the position of a device at regular time intervals. Note that LBACS components may be distributed across different physical spaces (e.g., office spaces and hardware/software components may be located in different buildings).

The correctness and security of the operation of LBACS depends on several conditions including, for example, the *liveness* of daemons in the mobile devices, the *confidentiality* of signals sent from daemons to the location server, and the *availability* of the location server. Such conditions need to be monitored at runtime, as attacks to LBACS components can compromise the whole system operation. However, as the components of LBACS are autonomous, runtime checks need to be based on some external monitoring entity receiving and checking events emitted from LBACS components. Furthermore, to ensure that the results of the monitoring process are correct, it is necessary to assess whether the runtime events received by the monitor are trustworthy, faulty, or the result of an attack onto the system. Delayed or dropped signals from the daemons of mobile devices would, for instance, prevent the detection of the location of a device and lead to not allowing it to access certain resources. Also, an attack resulting in delaying or dropping the signals of the location server can compromise the whole operation of LBACS, preventing access to any resource controlled by it.

As discussed earlier, runtime monitoring has been supported by several approaches and systems. However, to the best of our knowledge, the problem of assessing the trustworthiness of events that underpin the monitoring process, in cases of cyber systems like LBACS, has received less attention in the literature.

In this paper we present an approach that we have developed to address this problem. Our approach is based on attempting to generate possible *explanations* of the runtime events received by a monitor and checking whether these explanations are supported/confirmed by further operational evidence (i.e., other events that have been generated by the system under surveillance and could be the consequence of the same explanations). The generation of explanations of runtime events is based on hypothesising the possible causes of events using a *diagnostic model* about the operation of the system under surveillance and its components.

The process of generating explanatory hypotheses is based on *abductive reasoning* [30]. Following the generation of hypothetical *explanations* for an event, we assess their validity by identifying the *effects* that the generated explanations would have if they were correct, and checking whether these effects correspond and can, therefore, be confirmed by other runtime events in the monitor's log.

It should be noted that the exploration of violations of desired system properties has been the focus of work in software engineering (e.g., [6][15][31][37]) and AI (e.g., [11][32]) and has been often termed as "diagnosis". In this work, diagnosis is typically concerned with the detection of the root causes of violations or faults through the identification of the trajectories of events that have caused them. This is similar to the work that we present in this paper. However, the focus of our approach is different from existing work on diagnosis, since our focus is the assessment of the trustworthiness of the events that indicate the presence of violations of desired system properties as opposed to the identification of the root cause of a detected system fault or security violation (see Sect VI for a more detailed comparison).

To cope with inherent uncertainties that arise in assessing the trustworthiness of monitoring events, the assessment of the validity of the explanations generated for the events and the event trustworthiness itself is based on the computation of beliefs using belief measuring functions grounded in the Dempster-Shafer (DS) theory of evidence [33]. The belief-based assessment of event trustworthiness becomes what we term in our approach as "event genuineness".

To test and validate our approach, we have developed an *event assessment module* as part of the *Event Reasoning Toolkit (EVEREST* [35][36]*)*. EVEREST has been developed to support the monitoring of security and dependability properties for distributed systems expressed as monitoring rules in a formal temporal logic language that is based on Event Calculus (EC) [22], called *EC-Assertion*.

Early versions of our event assessment approach have been presented in [39][40][41]. The main contributions of this paper with respect to our earlier work are related to:

(a) the introduction new belief functions for assessing event genuineness and the formalisation of these functions in the context of the DS theory of evidence;
(b) the introduction of the algorithm for generating hypothetical explanations of events;
(c) the explanation of how the results of the diagnosis process can be utilised in monitoring; and
(d) the experimental evaluation of our approach.

The rest of this paper is organised as follows. Section II provides background information regarding the EVEREST framework that underpins the implementation of our approach. Section III presents the process of generating and validating event explanations. Section IV describes the belief functions used to assess event genuineness. Section V presents an experimental evaluation of our approach. Section VI overviews related work. Finally, Section VII provides some concluding remarks about our approach and outlines plans for further work.

II. PRELIMINARIES: THE EVEREST FRAMEWORK

Before discussing the details of our approach, we provide an overview of the EVEREST system and the way in which it expresses the properties to be monitored. This is necessary in order to understand the form of the diagnosis model used by our approach.

A. Overview of EVEREST

Fig. 2 shows the overall architecture of EVEREST, which consists of an *Event Collector*, a *Monitor*, a *Predictive Monitoring Module*, a *Manager* and a *Diagnosis Module* (i.e., the module implementing the approach that we describe in this paper).

The event collector receives notifications of events that are captured by external captors, transforms these events into an internal representation used by EVEREST, stores the transformed events into an internal event database (aka monitor log), and notifies other EVEREST components that events have occurred.

The monitor checks if the received events violate specific properties. This module is implemented as a reasoning engine whose operation is driven by the monitoring specification provided for a system. This specification consists of *monitoring rules* expressing the properties that need to be checked at runtime and *monitoring assumptions* that are used to set and update the values of state variables required for monitoring. Monitoring rules and assumptions are expressed as *EC-Assertion* formulas as discussed in Section II.B.

The predictive monitoring module of EVEREST receives the same stream of events as the monitor but checks for potential (as opposed to definite) violations of monitoring rules. Both the monitor and the predictive monitoring module take into account assessments of the genuineness of runtime events, if this is required according to the monitoring specification.

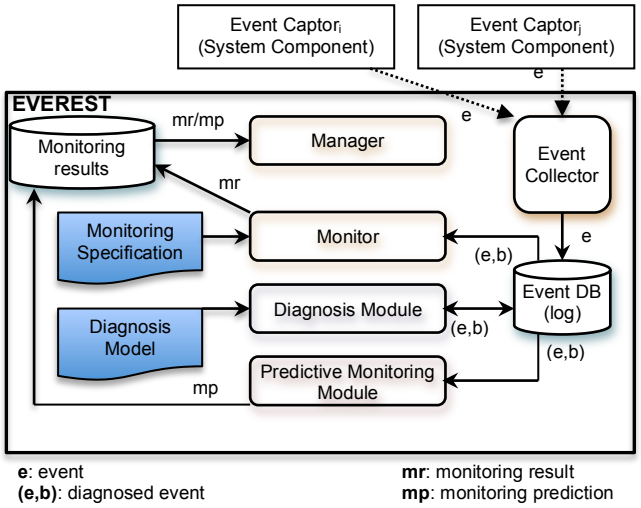

Figure 2. EVEREST architecture

The assessment of the genuineness of runtime events is the responsibility of the diagnosis module of EVEREST. The module generates explanations of runtime events based on a *diagnosis model* of the system under surveillance and uses them to compute beliefs in the trustworthiness of events. These beliefs are then stored in the event database (*Event DB*) from which they can be accessed by the monitor and the predictive monitoring module. The diagnosis model used in this process contains assumptions about the operation of the system under surveillance. These assumptions take the form of sequences of operation invocations and responses as well as the effect that these actions have in the state of the system (see Sect III.B below for examples).

The monitor and the predictive monitoring module store their results in a monitoring results database (*Monitoring Results DB*), along with diagnostic information regarding the genuineness of the events that have caused a violation or underpin the prediction about a violation. This database is accessed by the *Manager,* which provides an interface to external clients for retrieving detected violations for a given system.

B. Monitoring specifications and diagnostic models

In EVEREST the properties to be monitored are expressed as formulas of *EC-Assertion* [35]. *EC-Assertion* is a restricted form of the first order temporal logic language of Event Calculus (EC) [22].

The basic modeling elements of EC are *events* and *fluents*. An event is something that occurs at a specific instance of time and is of instantaneous duration. The occurrence of an event at a particular instance of time is expressed by the predicate *Happens*. Events may change the state of the system that is being modeled. More specifically, an event can initiate or terminate a state. States are modeled as fluents and the initiation and termination of a fluent by an event are expressed by the predicates *Initiates* and *Terminates*, respectively. In addition to the predicates *Happens*, *Initiates* and *Terminates*, EC offers the predicate *HoldsAt* which expresses a check of whether a given fluent holds at a specific instance of time.

TABLE I. EC-ASSERTION PREDICATES & THEIR MEANINGS

Predicate	**Meaning**
Happens(e,t$\Re(t^L,t^U)$)	An event *e* of instantaneous durations occurs at some time point *t* within the time range $\Re(t^L,t^U)$ ($\Re(t^L,t^U)=[t^L,t^U]$).
HoldsAt(f,t)	A state (aka *fluent*) *f* holds at time *t*. This is a derived predicate that is true if the *f* has been initiated by some event at some time point t' before t and has not been terminated by any other event within the range [t',t].
Initiates(e,f,t)	Fluent *f* is initiated by an event *e* at time *t*
Terminates(e,f,t)	Fluent *f* is initiated by an event *e* at time *t*
Initially(f)	Fluent *f* holds at the start of system operation.
<rel>(x,y) *where <rel>::= = | < | > | ≤ | ≥ | ≠*	The relation <rel> holds between the x and y.

EC-Assertion adopts the basic modeling constructs of EC but introduces special terms to represent types of events, fluents and computations needed for runtime monitoring, and special predicates to express comparisons between basic data type values (see [35] for more details), and terms denoting calls to special *built-in functions* that are available in order to compute complex computations (e.g. to compute statistical

information and fit statistical distributions over data). The exact syntactic form and meaning of the predicates *Happens*, *Initiates*, *Terminates*, *HoldsAt* and *Initially* in *EC-Assertion* are shown in Table I.

The properties monitored by EVEREST are expressed in *EC-Assertion* as *monitoring rules* of the form

$$mr: B_1 \wedge \ldots \wedge B_n \Rightarrow H_1 \wedge \ldots \wedge H_m.$$

In this form, the predicates B_i and H_j are restricted to *Happens*, *HoldsAt* and *relational* predicates. A monitoring rule *mr* is satisfied if when the sub formula $B_1 \wedge \ldots \wedge B_n$ (aka *body* of the rule) becomes *True*, the sub formula $H_1 \wedge \ldots \wedge H_m$ (aka *head* of the rule) also becomes *True*.

An example of an EVEREST monitoring rule is given below:

Monitoring Rule R1[1]

```
Happens(e(_eID1,_devID,_locServerID, REQ, signal(_devID),_locServerID), t1,
R(t1,t1)) ⇒
Happens(e(_eID2,_devID,_locServerID, REQ, signal(_devID),_locServerID), t2,
R(t1,t1+m)) ∧ _eID1 ≠ _eID2
```

This rule refers to the LBACS system introduced in Sect. I and checks the *liveness* of daemons in the mobile devices accessing LBACS. In particular, R1 checks if the location server of LBACS (signified by the value of the variable *_locServerID*) receives signals from a given device (*_devID*) every *m* time units. Every signal that is sent by a device and captured by the location server at some time point t_1 will trigger the check of the rule by matching the predicate `Happens(e(_eID1,_devID,_locServerID,REQ,signal(_devID),_locServerID),t1,R(t1,t1))` in the body of the rule. Subsequently, the monitor of EVEREST will expect to receive another signal (event) from the same device that could be matched with the predicate `Happens(e(_eID2,_devID, _locServerID,REQ,signal(_devID),_locServerID),t2,R(t1,t1+m))` at some time point t_2 that is within the range $[t_1, t_1+m]$ and if no such a signal is received the rule will be violated.

Another example of a monitoring rule of LBACS is:

Monitoring Rule R2:

```
Happens(e(_eID1,_devID,_ctrlServerID, REQ,
requestAccess(_devID,_result),_ctrlServerID), t1,R(t1,t1)) ⇒
HoldsAt(AUTHENTICATED(_devID),t1)
```

This rule checks if a device has been authenticated when it makes a request to the control server for accessing a resource. The authentication of a device that sends the request is indicated by the fluent `AUTHENTICATED(_devID)` and the rule checks the authentication through the predicate `HoldsAt(AUTHENTICATED(_devID),t1)`. As discussed earlier, however, *HoldsAt* is a derived predicate that is true only if its fluent argument has been initiated by an event prior to the time at which the predicate is checked and has not been terminated by any other event until then. Thus, when the monitor receives a resource access request event matching the *Happens* predicate in its body (i.e., a `requestAccess(_devID,_result)` event) at some time point t_1, it will need to check if the *HoldsAt* predicate in its head is true at the same time.

Since, in EC fluents can only be initiated and terminated by events, the monitoring specification in this instance will need to specify how the fluent `AUTHENTICATED(_devID)` is initiated and (if applicable) terminated. In EVEREST, this is specified by monitoring assumptions. A monitoring assumption in *EC-Assertion* has the form *:* $B_1 \wedge \ldots \wedge B_n \Rightarrow H_1 \wedge \ldots \wedge H_m$ where predicates B_i are restricted to *Happens*, *HoldsAt* and *relational* predicates and the predicates H_j are restricted to *Initiates* and *Terminates* predicates.

In the case of *R2* the monitoring assumption used to establish how `AUTHENTICATED(_devID)` is initiated is:

Monitoring Assumption MA1

```
Happens(e(_eID2,_ntwrkCntrl,_devID, RES, connect(_devID,
_connection),_ntwrkCntrl), t1, R(t1,t1)) ∧ (_connection ≠ NIL) ⇒
Initiates( e(_eID2, …), AUTHENTICATED(_devID), t1)
```

According to this assumption, a device is authenticated if it has made a request for connection to the controller of the network in the area covered of LBACS at least once (see the event `connect(_devID, _connection)` and this request has been accepted (as specified by `_connection ≠ NIL`).

It should be noted that the general form of monitoring assumptions of EVEREST that was shown above allows the expression of more complex patterns of fluent initiation and termination than the pattern shown in

[1] In all *EC-Assertion* formulae of this paper, the names of non time variables start with underscore (i.e., _varName) and time variable names are always "t" with a subscript (i.e., t_i).

the previous example. The monitoring assumption MA2 below gives an example of a fluent that is used to record movements of a device in different parts of the network of the LBACS case study (i.e., the fluent `MOVEMENT(_devID,_originalNetworkController,_destinationNetworkController)`). As indicated by MA2, when a device makes two consecutive requests for connection to two different areas of the LBACS network, the fluent `MOVEMENT` is initiated to indicate the movement of the device between these two areas.

Monitoring Assumption MA2

$$\begin{aligned}&\text{Happens}(e(\_eID1,\_ntwrkCntrl1,\_devID, REQ, connect(\_devID,\\&\_connection1),\_ntwrkCntrl1), t_1, R(t_1,t_1)) \wedge\\&\text{Happens}(e(\_eID2,\_ntwrkCntrl2,\_devID, REQ, connect(\_devID,\\&\_connection2),\_ntwrkCntrl2), t_2, R(t_1,t_2)) \wedge\\&(\_ntwrkCntrl1 \neq \_ntwrkCntrl2) \wedge\\&\neg\exists t_3: \text{Happens}(e(\_eID3,\_ntwrkCntrl3,\_devID, REQ, connect(\_devID,\\&\_connection3),\_ntwrkCntrl3), t_3, R(t_3,t_3)) \wedge (t_3 \leq t_1) \wedge (t_3 \leq t_2) \Rightarrow\\&\text{Initiates}(e(\_eID2, \ldots), \text{MOVEMENT}(\_devID,\_ntwrkCntrl1,\_ntwrkCntrl2),t_2)\end{aligned}$$

To assess the trustworthiness of events, EVEREST relies on a diagnostic model (see Fig. 1). This model consists of *diagnostic assumptions* about the system under surveillance.

Diagnostic assumptions are specified as *EC-Assertion* formulas of the form $a: B_1 \wedge \ldots \wedge B_n \Rightarrow H$. In this form, B_i can be any of the EC predicates shown in Table I, and H can be a *Happens*, *Initiates* or *Terminates* predicate. *HoldsAt* predicates cannot be specified in the head (H) of diagnostic assumptions since their truth-value can only be derived from the axioms of EC. Diagnostic assumptions enable the specification of expected sequences of events, which may be *observable events* (i.e., events of the system under surveillance that are visible to the monitor) or *internal events* of the system under surveillance. Internal events are events that might not be desirable to emit externally (e.g. due to performance or confidentiality reasons) but are often necessary to include in a diagnostic model in order to indicate how a system operates internally. The following formula gives an example of a diagnostic assumption for LBACS:

Diagnostic Assumption DA1

$$\begin{aligned}&\text{Happens}(e(\_eID1,\_x1,\_x2, activeDaemon(\_devID),\_x3),t_1,R(t_1,t_1)) \Rightarrow\\&(\exists t_2{:}\text{Time})\text{Happens}(e(\_eID2,\_devID,\_locSerID, signal(\_devID),\_locSerID), t_2,\\&R(t_1,t_1+2))\end{aligned}$$

DA1 states that if the daemon of a device is active at a given time point t_1, then it must send a signal to the location server within 2 time units. As we discuss in Sect. IV below, this assumption enables the diagnosis process to create a hypothesis (explanation) that the daemon of a device, which appears to have sent a signal to the location server of LBACS, was active for a given period before the dispatch of the signal.

III. Assessment of Event Genuineness

A. Overview

The event assessment process has three phases: (i) the *explanation generation* phase in which hypotheses about the possible explanations of events are generated; (2) the *effect identification* phase in which the possible consequences (effects) of the hypothetical explanations of an event are derived; and (3) the *explanation validation* phase in which the expected consequences of each explanation are checked against the event log of the monitor to establish if there are events that match the consequences and, therefore, provide supportive evidence for the validity of the explanation.

The above phases are described in detail below.

B. Generation of event explanations

The first phase of the assessment process uses abductive reasoning to find possible explanations for events. The algorithm used to carry out this process is shown in Fig. 3.

The algorithm gets as input an atomic predicate e for which an explanation is required and the boundaries of the time range of this predicate $t_{min}(e)$ and $t_{max}(e)$. It also has a fourth input parameter, called f_{init}, representing the initial formula that is to be used for generating explanations. This parameter is not used in the initial invocation of *Explain* since the objective of the process is to find all the possible alternative explanations of the input predicate. In subsequent recursive invocations of the algorithm, however, it is used to indicate the identifier of the last formula that was used in the generation of an on-going explanation since, along with explanations, the algorithm records the backward chaining path through which the abduced atomic predicates of each explanation were generated (see lines 10 and 28 in Fig 3). Given an input predicate e that is to be explained, if the predicate symbol of e is an abducible predicate (i.e., a predicate whose validity

can be assumed through the abduction process), a pair of the predicate *e* and its time range (i.e., *(e, $t_{min}(e)$, $t_{max}(e)$)*) is added to the current list of explanations Φ_e and the algorithm terminates by returning Φ_e (see lines 8 and 44 in Fig. 3).

```
Explain(e, t_min(e), t_max(e), f_init, Φ_e)
 1.  // IN: e: an event and/or grounded atomic predicate to explain
 2.  // IN: t_min(e), t_max(e): min and max boundaries of the time range of e
 3.  // IN: f_init: set of explanations generated in the context of explaining e
 4.  // OUT: Φ_e: a list of explanations of atomic predicate e
 5.  Φ_e = [ ]_OR
 6.  // e is an abducible atom; add e to the current explanation
 7.  If e ∈ ABD Then
 8.      Φ_e = append(Φ_e , [(e, t_min(e), t_max(e))])
 9.      //APath_e[ ]: list of identifiers of f visited in abducing e
10.      append(APath_e[ ], f_init)
11.  // e is not an abducible atom; find explanations for it
12.  Else
13.   For all f ∈ AS Do //search for explanations based on all f in AS
14.          // mgu(f,e): most general unifier of event e and predicate p
15.       u = mgu(head(f), e)
16.       If u ≠ ∅ & u covers non time variables in body(f) then
17.        //CND_f: list of predicates in body of f (conditions)
18.       Copy body(f) into CND_f
19.       Failed = False
20.       Φ_f = []_AND //Φ_f : list of explaining elements of CND_f
21.       While Failed = False and CND_f ≠ ∅ DO
22.          select C from CND_f
23.          ComputeTimeRange(C, t_min(e), t_max(e), t_min(C), t_max(C))
24.          If t_min(H) ≠ NULL and t_max(C) ≠ NULL Then
25.             C_u = ApplyUnification(u, C)
26.          If C_u ∈ ABD Then // C_u is an abducible atom
27.             Φ_f = append (Φ_f , [(C_u, t_min(C), t_max(C))]_ABD )
28.             append(APath_Cu[ ], f)
29.          Else    // C is not an abducible atom
30.                find a derived predicate or event e_c such that: mgu(e_c, C_u ) ≠ ∅ & t_min(e_c)
     ≥ t_min(C) & t(e_c) ≤ t_max(C)
31.             // no recorded/derived predicate matching C
32.             If e_c ≠ NULL Then
33.                Φ_C = Explain (C, t_min(C), t_max(C), f)
34.                If Φ_C is empty Then Failed = True
35.                Else Φ_f = append(Φ_f , Φ_C) End If
36.             End If
37.          End If
38.       End If
39.    End While
40.    If Failed = False Then Φ_e = append (Φ_e,Φ_f) End if
41.     End if
42.   End For
43.  End If
44.  return(Φ_e)
   END Explain
```

Figure 3. Explanation generation algorithm

If *e* is not an abducible predicate, however, *Explain* searches through the diagnosis model of the system under surveillance (i.e., the set of diagnostic assumptions AS for the system) to identify if there are any diagnostic assumptions that could explain *e*, i.e., assumptions that have a predicate *P* in their head, which can be unified with *e* (see line 13). For every assumption *f* that has such a predicate, the algorithm checks if a unifier of *P* and *e* exists, and, if it exists, whether the unifier provides bindings for all the non time variables in the body of *f* (see lines 15-16 in the algorithm). The unification test is performed by calling the function *mgu(head(f),e)*. This function returns the most general unifier between its input formulas [20] for all but the non-time variables of these formulas.

The exclusion of time variables from the unification test is because for time variables, the explanation process needs to find if there are feasible time ranges for them rather than merely checking variable unifiability. More specifically, if all other unifiability conditions are satisfied, the algorithm checks if the time constraints imposed by the event *e* on each of the instantiated predicates in the body of *f*, can lead to concrete and feasible time ranges for the predicate. To check this, all the constraints involving the time variable of the predicate *C* in the body of *f* (i.e., $t_{min}(C)$ and $t_{max}(C)$) are retrieved and t_C is replaced by the time stamp of *e*.

Subsequently, the boundaries of the possible values for the time variables of the other predicates in the body of *f* are computed (see the call of *ComputeTimeRange(C, $t_{min}(e)$, $t_{max}(e)$, $t_{min}(C)$, $t_{max}(C)$)* in line 23 of the algorithm). *ComputeTimeRange* treats this computation as a linear programming problem due to the way in which constraints for the time variables of *EC-Assertion* formulas are specified. More specifically, each *EC-*

Assertion formula must define an upper and a lower boundary for the time variables of all its predicates. Given the time variable t_k of a predicate P_k in a formula *f*, its upper and lower boundaries *ub* and *lb* are defined by linear expressions of the form: $lb = l_0 + l_1 t_1 + l_2 t_2 + ... + l_n t_n$ and $ub = u_0 + u_1 t_1 + u_2 t_2 + ... + u_n t_n$ where t_i (i=1, ..., n) are the time variables of the remaining predicates in *f* and the constraints related to t_k are of the form:

$l_0 + l_1 t_1 + l_2 t_2 + \ldots + l_n t_n \leq T_E$ (C1)

$T_E \leq u_0 + u_1 t_1 + u_2 t_2 + \ldots + u_n t_n$ (C2)

From (C1) and (C2), it may be possible to compute the minimum and maximum possible values of any variable t_i (i=1, ..., n) in the predicates by solving the linear optimisation problems $max(t_i)$ and $min(t_i)$ subject to constraints *C1* and *C2*. By solving these optimization problems for each of the time variables of the predicates in the body of a diagnostic assumption *f*, it can be established if a concrete and feasible time range exists for the time variables of *f*.

If such a range exists, the explanation generation algorithm applies the unifier of *e* and *C* to the predicates in the body of *f* (see line 25 in the algorithm). Then it checks if the instantiated predicates in the body of *f* are abducible predicates. When this happens, the instantiated abducible predicates in the body of *f* are added to the partially developed explanation (see lines 27-28 in the algorithm). If an instantiated predicate in the body of *f* is not an abducible predicate but can be unified with an event already recorded in the monitor's log or with a predicate that could be derived by a diagnostic assumption (derivable predicate), the algorithm tries to find an explanation for the predicate recursively (see lines 30-37 in the algorithm).

If an instantiated predicate in the body of *f* is neither an abducible predicate nor does it correspond to a recorded event or derivable predicate, the explanation generation process searches for other diagnostic assumptions whose head predicate can be unified with *e*. When an assumption *a* that underpins a partially generated explanation has some predicate B_j in its body that is not an abducible, does not match with any logged event, and cannot be explained through other assumptions, *a* is skipped and no further attempt is made to generate an explanation from it (see line 34 in the algorithm).

As an example of applying *Explain,* consider the search for an explanation of an event *E1=Happens(e(E1,D33,LS1,signal(D33),LS1),15,R(15,15))*. This event represents a signal sent to the location server of LBACS from a device D33. Assuming the diagnostic assumption *DA1* introduced in Section II.B, the diagnostic process detects that *E1* can be unified with the predicate in the head of *DA1* (the unifier of the two predicates is *U={_eID2/E1, _devID/D33, _locSerID/LS1,t2/15}*).

Following this unification, the linear constraint system that will be generated for the time variable t_1 in *DA1* will include the constraints $t_1 \leq 15$ and $15 \leq t_1 + 2$ or, equivalently, $15 - 2 \leq t_1$ and $15 \leq t_1$. Thus, a feasible time range exists for t_1 (i.e., $t1 \in [13,15]$) and, as the non time variables in the body of *DA1* are covered by U, the conditions of the explanation generation process are satisfied and the formula *Ê:Happens(e(eID1, _x1, _x2, activeDaemon(D33), _x3), t1, R(13, 15)* will be generated as a possible explanation of *E1*. The meaning of this explanation is that the daemon of *D33* had been active in the period of up to 2 time units before the dispatch of the signal and sent the signal (i.e., a benign explanation for the dispatch of the signal).

C. *Identification of possible explanation consequences*

Following the generation of an explanation $\hat{E}$ for a logged event *e*, the assessment process identifies all the consequences other than *e* that $\hat{E}$ would have and uses them to assess the validity of $\hat{E}$. This reflects the hypothesis that, if the additional consequences of $\hat{E}$ also match with events in the log, then there is further evidence that $\hat{E}$ is likely to be true and the cause of *e* (as well as the additional events matching its further consequences).

The identification of explanation consequences is based on the diagnostic model of the system and deductive reasoning. Given an explanation $\hat{E} = P_1 \wedge \ldots \wedge P_n$ where P_i (i=1,…,n) are abduced atomic predicates, the diagnosis process iterates over the predicates P_i and, for each of them, it finds diagnostic assumptions $a: B_1 \wedge \ldots \wedge B_n \Rightarrow H$ having a predicate B_j in their body that can be unified with P_i. For each of these assumptions, the process checks if the rest of the predicates in its body match with conjuncts of $\hat{E}$ or an event in the log. In cases where these conditions are satisfied, if the predicate *H* in the head of the assumption is fully instantiated and the boundaries of its time range are determined; *H* is derived as a possible consequence of P_i. Then, if *H* is an observable predicate, i.e., a predicate that referring to an observable event, and can be matched with an event in the current log, *H* is added to the possible effects of $\hat{E}$. Otherwise, if *H* is not an observable predicate, the diagnosis process tries to generate the consequences of *H* recursively and, if it finds any such consequences that correspond to observable predicates, it adds them to the set of the consequences of $\hat{E}$.

As an example, consider the explanation $\hat{E}$ derived in Section III.B above for the LBACS. A possible consequence of this explanation is

C:Happens(e(_eID2,D33,LS1,signal(D33),LS1),t2,R(13,17))

This consequence is derived from $\hat{E}$ and *DA1* and means that any signal sent by *D33,* other than the one represented by the event *E1*, within the time range [13,17] would support the validity of $\hat{E}$.

Following the identification of the expected consequences of an explanation, the assessment process searches for events in the log that can be unified with them and, if it finds any, it assesses the genuineness of the matched events.

It should be noted that a consequence is a non-ground event e_i that is expected to be produced by an event captor *c* within a time range $[t_i^L, t_i^U]$. Hence, to match an event e_{log} in the log with e_i, three conditions should be satisfied:

- e_{log} should be produced by the same event captor as e_i, *(CND1)*
- e_{log} should be unifiable with e_i (*CND2*), and
- the timestamp of e_{log} should be within the time range of e_i (i.e., $t_i^L \leq t_{log} \leq t_i^U$) (*CND3*).

IV. Beliefs in Event Genuineness & Explanation Validity

A. Uncertainty in the assessment process

The assessment of the trustworthiness of events based on the process we described in Section III has three uncertainties.

The first of these uncertainties relates to the generation of explanations through abductive reasoning (AR). Given a known causal relation (C ⇒ E) between a cause (C) and an effect (E), AR assumes that C is true every time that E occurs. This is not, however, always the case (or otherwise it would also be known that E ⇒ C). The second uncertainty relates to the possibility of having alternative explanations for the same effect. In cases where it is known, for example, that C'⇒ E, there is uncertainty as to which of C or C' should be assumed to have been the cause of E when E occurs.

The third uncertainty relates to the fact that the absence of an event from the monitor's log does not necessarily imply that the event has not occurred. This possibility affects the explanation validation phase since the absence of an event matching a consequence of an explanation, at the time of the search, does not necessarily mean that such an event has not occurred.

More specifically, as we discussed in Sect. IV.C, to match an event in the log of the monitor with the consequence of an explanation conditions *CND1—CND3* should be satisfied. However, there may be cases where an event, which would satisfy these conditions, might have occurred but not arrived at the monitor yet, due to communication delays in the "channel" between the monitor and the event's captor.

To cope with these uncertainties, we use an approximate assessment of the validity of explanations and event genuineness, based on belief functions grounded in the DS theory. The use of DS beliefs, instead of classic probabilities, enables us to represent uncertainty when the existing evidence cannot directly support both the presence and the absence of events. In the case of validating an explanation consequence e_i that should be matched by an event occurring up to the time point t_i^U, for example, if the timestamp t' of the latest event from the captor of e_i is less than t_i^U, there is no evidence for the existence or absence of an event matching e_i. Also, if $t'>t_i^U$ whilst the presence of genuine events with a timestamp in the range $[t_i^U,t']$ does provide evidence that an event matching e_i has not occurred, the absence of such events does not provide evidence that e_i has occurred.

A possible alternative to the use of DS theory would be the use of a Bayesian approach in our framework [16]. A Bayesian approach, however, would require the identification and assignment of a priori probabilities to each of the possible diagnostic assumptions in advance, as well as, conditional probabilities for the consequences of these assumptions. Besides this additional upfront cost, a Bayesian approach would need to address issues like the accuracy all the a priori and conditional probabilities of diagnostic assumptions.

Due to these reasons, we use DS belief functions to represent the uncertainty in the assessment of event genuineness. These belief functions are introduced below following a short overview of the different types of belief functions in DS theory.

B. DS preliminaries

In DS theory, there are two types of belief functions: (i) *basic probability assignment functions* and (ii) *belief functions*. Both these types of functions must be defined over a set θ of mutually exclusive propositions representing exhaustively the phenomena to which beliefs should be assigned. The set θ is called *frame of discernment (FoD)* in the DS theory.

A basic probability assignment (BPA) is a function *m* that provides a measure of belief to the truth of a subset *P* of θ that cannot be split to any of the subsets of *P* (*P* is taken to express the logical disjunction of the propositions in it). Formally, a BPA is defined as a function preserving the following axioms:

$$m: \wp(\theta) \rightarrow [0,1] \qquad (a1)$$

$m(\varnothing) = 0$ (a2)

$\Sigma_{P \subseteq \wp} m(P) = 1$ (a3)

where $\wp(\theta)$ denotes the power set of θ.

The subsets *P* of θ for which $m(P) > 0$ are called "focals" of *m* and the union of these subsets is called "core" of *m*. Each basic probability assignment *m* induces a unique belief function *Bel* that is defined as:

$Bel: \wp(\theta) \rightarrow [0,1]$ (a4)

$Bel(A) = \Sigma_{B \subseteq A} m(B)$ (a5)

Bel measures the total belief committed to the set of propositions *P* by accumulating the beliefs committed by the BPA underpinning it to the subsets of *P*. *Bel* must preserve the following axioms:

$Bel(\varnothing) = 0$ (a6)

$Bel(\theta) =$ (a7)

$\Sigma_{I \subseteq \{1,\ldots,n\} \text{ and } I \neq \varnothing} (-1)^{|I|+1} Bel(\cap_{i \in I} P_i) \leq Bel(\cup_{i=1,\ldots,n} P_i)$

where $n = |\wp(\theta)|$ and $P \subseteq \theta$, $(i=1,\ldots,n)$ (a8)

Finally, in DS theory two BPAs – say m_1 and m_2 – that have been defined over the same FoD can be combined according to the rule of the "orthogonal sum":

$m_1 \oplus m_2 (P) = (\Sigma_{X \cap Y = P} m_1(X) \times m_2(Y)) / (1 - k_0)$

where $k_0 = \Sigma_{V \cap W = \varnothing \text{ and } V \subseteq \wp \text{ and } W \subseteq \wp} m_1(V) \times m_2(W)$ (a9)

C. Belief in event genuineness

The computation of a BPA and a belief in event genuineness needs to cover two types of events, namely runtime *events* recorded in the monitor's log, and *consequent events* generated as expected consequences of explanations of other runtime events. The genuineness of events of the latter type must be assessed since it underpins the assessment of the validity of explanations (see Section III.C).

The BPA to the genuineness of an event is defined as follows:

Definition 1: The BPA function the genuineness of an event *e*, expected to occur within $[t_e^L, t_e^U]$, is defined as follows:

(a) if $U(e) \neq \varnothing$:

$$m^G(P) = \begin{cases} m^{EX}(\bigvee_{e_i \in U(e)} E^{e_i,U_o,W}) & \text{if } P = G^{e,U_o,W} \\ m^{EX}(\bigwedge_{e_i \in U(e)} \neg E^{e_i,U_o,W}) & \text{if } P = \neg G^{e,U_o,W} \\ 1 - (m^{EX}(\bigvee_{e_i \in U(e)} E^{e_i,U_o,W}) + m^{EX}(\bigwedge_{e_i \in U(e)} \neg E^{e_i,U_o,W})) & \text{if } P = G^{e,U_o,W} \vee \neg G^{e,U_o,W} \end{cases}$$

(b) if $U(e) = \varnothing$

$$m^G(P) = \begin{cases} 0 & \text{if } t_c^{max} < t_e^U, \text{ and } P = \neg G^{e,U_o,W} \text{ or } P = G^{e,U_o,W} \\ 0 & \text{if } t_c^{max} \geq t_e^U \text{ and } P = G^{e,U_o,W} \\ m^{EX}(\bigvee_{e_i \in A(e)} E^{e_i,U_o,W}) & \text{if } t_c^{max} \geq t_e^U \text{ and } P = \neg G^{e,U_o,W} \\ 1 - m^{EX}(\bigvee_{e_i \in A(e)} E^{e_i,U_o,W}) & \text{if } t_c^{max} \geq t_e^U \text{ and } P = G^{e,U_o,W} \vee \neg G^{e,U_o,W} \end{cases}$$

where

- $G^{e,U_o,W}$ and $\neg G^{e,U_o,W}$ are propositions denoting that *e* is genuine and not genuine, respectively[2];
- *U(e)* is the set of concrete events in the monitor's log that can be unified with *e*;
- t_c^{max} is the latest timestamp of all the events in the log that have been received from the captor *c* of *e*;
- *A(e)* is the set of concrete events in the log that have been received from the captor *c* of *e* but occurred after *e* (i.e., events having a timestamp *t'* such that $t' \geq t_e^U$);
- $E^{e_i,U_o,W}$ and $\neg E^{e_i,U_o,W}$ are propositions denoting that, without considering the events in the set U_o, a concrete event e_i, which can be unified with *e*, has a valid explanation or no valid explanation, respectively, within the time range $[t_{mid}-w/2, t_{mid}+w/2]$ (t_{mid} is the midpoint of the range $[t_e^L, t_e^U]$); and
- m^{EX} is the BPA function resulting from the combination of the BPAs m_i^{EX} to the propositions $E^{e_i,U_o,W}$ and $\neg E^{e_i,U_o,W}$ that are defined in Definition 2 (i=1,…,n; n=|U(e)|); i.e., $m^{EX} = m_1^{EX} \oplus m_2^{EX} \oplus \ldots \oplus m_n^{EX}$. ❑

[2] All propositions used in Definitions 1-3 are defined in reference to a common frame of discernment that has been formally defined in [38].

Definition 1 distinguishes between two cases: (a) the case of events that can be matched with runtime events in the monitor's log, i.e., when U(e)≠∅, and (b) the case of events that cannot be matched with any event in the monitor's log U(e)=∅. Case (a) may involve both runtime and consequent events. For a runtime event *e*, U(e)={e}. For a consequent event *e*, which may be expected to occur within a given time range (as opposed to a single time point) and/or have parametric arguments, U(e) may contain more than one runtime events matching *e*.

If U(e)≠∅, the criterion that underpins Definition 1 is that an event *e* is considered as genuine if any of the runtime events matching it has at least one valid explanation (see disjunction $\bigvee_{e_i \in U(e)} E^{e_i,U_o,W}$). Also an event is considered to be non-genuine if none of the events matching it has a valid explanation (see conjunction $\bigwedge_{e_i \in U(e)} \neg E^{e_i,U_o,W}$). Consequently, m^{GN} measures the BPA to the genuineness and non genuineness of an event as the BPA $m^{EX}(\bigvee_{e_i \in U(e)} E^{e_i,U_o,W})$ and the BPA $m^{EX}(\bigwedge_{e_i \in U(e)} \neg E^{e_i,U_o,W})$, respectively.

For consequent events *e* that cannot be unified with any event in the log, i.e., when U(e)=∅, the computation of m^{GN} depends on the latest timestamp of the captor of *e*, t_c^{max}. If $t_c^{max} < t_e^U$, the occurrence of an event matching *e* cannot be precluded. In such cases, m^{GN} remains agnostic by assigning a zero basic probability to $G^{e,U_o,W}$ and $\neg G^{e,U_o,W}$. If $t_c^{max} \geq t_e^U$, m^{GN} assigns a zero basic probability to the genuineness of *e* because there is no event matching it. However, since t_c^{max} is determined by the time stamp of the latest event received from captor *c*, its accuracy would also need to be assessed. This is because if the event with timestamp t_c^{max} is not genuine, t_c^{max} will not be correct. To address this possibility, m^{GN} computes the basic probability of the non genuineness of the event of interest (i.e., $\neg G^{e,U_o,W}$) as the combined basic probability of the genuineness of the events in the log that have occurred after t_e^U (i.e., the basic probability assigned to the disjunction $\bigvee_{e_i \in A(e)} E^{e_i,U_o,W}$). This is because if all the events in *A(e)* are confirmed by valid explanations, then captor *c*'s time is more likely to have progressed beyond t_e^U and, therefore, the absence of events matching *e* would be more certain. Fig. 4 presents the time ranges that underpin the formulation of sets U(e), A(e) and the proposition $E^{e_i,U_o,W}$.

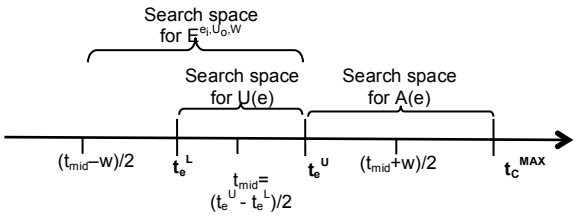

Figure 4. Time ranges for U(e), A(e) and $E^{e_i,U_o,W}$

The functional form of the BPA m^G and the belief function induced by it are derived by the following theorem.

Theorem 1: *Let U(e) and A(e) be non empty set of runtime events, $E^{e_i,U_o,W}$ be a proposition denoting that a concrete event e_i in U(e) or A(e) has a valid explanation within the time range $[t_{mid}-w/2, t_{mid}+w/2]$, and $\neg E^{e_i,U_o,W}$ be a proposition denoting that e_i has no valid explanation within the time range $[t_{mid}-w/2, t_{mid}+w/2]$.*
(a) The BPA m^{EX} defined as $m^{EX} = m_1^{EX} \oplus m_2^{EX} \oplus \ldots \oplus m_n^{EX}$ where m_i^{EX} (i=1,…,n; n=|U(e)| or n=|A(e)|) are the BPA functions defined in Definition 2 has the following form:

$$m^G(x) = \begin{cases} \Pi_{i \in I} m_i^{EX}(E^{e_i,U_o,W}) \times \Pi_{j \in I'} m_j^{EX}(\neg E^{e_j,U_o,W}) & \text{if } x = \bigcap_{i \in I} E^{e_i,U_o,W} \bigcap_{j \in I'} \neg E^{e_j,U_o,W} \\ & \text{for all } I, I' \text{ such that } I \subseteq S \;\&\; I' = S - I \\ & \text{where } S=U(e) \text{ or } S=A(e)^3 \\ 0 & \text{if } x \neq \bigcap_{i \in I} E^{e_i,U_o,W} \bigcap_{j \in I'} \neg E^{e_j,U_o,W} \end{cases}$$

(b) The beliefs to the propositions $\bigvee_{e_i \in U(e)} E^{e_i,U_o,W}$, $\bigwedge_{e_i \in U(e)} \neg E^{e_i,U_o,W}$ and $\bigvee_{e_i \in A(e)} E^{e_i,U_o,W}$ produced by the belief function induced by m^G are computed by the following formulas:

$$Bel^G(\bigvee_{e_i \in U(e)} E^{e_i,U_o,W}) = \Sigma_{I \subseteq U(e) \text{ and } I \neq \emptyset} (-1)^{|I|+1} \{ \Pi_{i \in I} m_i^{EX}(E^{e_i,U_o,W}) \}$$

$$Bel^G(\bigwedge_{e_i \in U(e)} \neg E^{u,U_o,W}) = \Pi_{e_i \in U(e)} m_i^{EX}(\neg E^{e_i,U_o,W})$$

[3] For any two sets identified as I and I', I' denotes the complement set of I with respect to some common set S. Unless defined otherwise in a theorem or definition, I' denotes the complement set of I w.r.t the common frame discernment θ defined in [38].

$Bel^G(\bigvee_{e_i \subseteq A(e)} E^{e_i,U_o,W}) = \Sigma_{I \subseteq A(e) \text{ and } I \neq \emptyset}(-1)^{|I|+1}\{\Pi_{i \in I} m_i^{EX}(E^{e_i,U_o,W})\}$

The proof of Theorem 1 is given in [38]. Based on it, the belief in event genuineness is computed as:

(i) if $U(e) \neq \emptyset$

$$Bel^G(P) = \begin{cases} \Sigma_{I \subseteq U(e) \text{ and } I \neq \emptyset}(-1)^{|I|+1}\{\Pi_{i \in I} m_i^{EX}(E^{e_i,U_o,W})\} & \text{if } P = G^{e,U_o,W} \\ \Pi_{u \in U(e)} m_i^{EX}(\neg E^{e_i,U_o,W}) & \text{if } P = \neg G^{e,U_o,W} \end{cases}$$

(ii) if $U(e) = \emptyset$

$$Bel^G(P) = \begin{cases} 0 & \text{if } P = G^{e,U_o,W} \text{ or } P = \neg G^{e,U_o,W}, \text{ and } t_c^{max} < t_e^U \\ 0 & \text{if } P = G^{e,U_o,W} \text{ and } t_c^{max} \geq t_e^U \\ \Sigma_{I \subseteq A(e) \text{ and } I \neq \emptyset}(-1)^{|I|+1}\{\Pi_{i \in I} m_i^{EX}(E^{e_i,U_o,W})\} & \text{if } P = \neg G^{e,U_o,W} \text{ and } t_c^{max} \geq t_e^U \end{cases}$$

Bel^G depends on the BPA m_i^{EX} for computing the basic probability of the existence or not of a valid explanation for a concrete event e_i. The BPA function m_i^{EX} is defined as follows.

Definition 2: The BPA function m_i^{EX} for the existence of a valid explanation for a runtime event e_i is defined as:

(a) if $EXP(e_i) \neq \emptyset$:

$$m_i^{EX}(P) = \begin{cases} m^{VL}(\vee_{\hat{E}_j \in EXP(e_i)} VL(e_i,\hat{E}_j,U_o,W)) & \text{if } P = E^{e_i,U_o,W} \\ m^{VL}(\wedge_{\hat{E}_j \in EXP(e_i)}(\neg VL(e_i,\hat{E}_j,U_o,W)) & \text{if } P = \neg E^{e_i,U_o,W} \end{cases}$$

(b) if $EXP(e_i) = \emptyset$

$$m_i^{EX}(P) = \begin{cases} \alpha_1 & \text{if } P = E^{e_i,U_o,W} \\ 1 - \alpha_1 & \text{if } P = \neg E^{e_i,U_o,W} \end{cases}$$

where,

- $EXP(e_i)$ is the set of possible explanations generated by abduction for e_i and $\hat{E}_j$ is an explanation in this set,
- $VL(e_i,\hat{E}_j,U_o,W)$ $(\neg VL(e_i,\hat{E}_j,U_o,W))$ is a proposition denoting that $\hat{E}_j$ is (not) a valid explanation for e_i given the events in the log within the range $[t_{mid}-w/2, t_{mid}+w/2]$ excluding those in U_o,
- α_1 is a parameter taking values in [0,1] determining the default basic probability of event genuineness in the case of events that have no explanation (α_1 should be set to 0 if such events need to be considered as non genuine events), and
- m^{VL} is the BPA function resulting from the combination of the basic probability assignments m_i^{VL} to the propositions $VL(e_i,\hat{E}_j,U_o,W)$ and $\neg VL(e_i,\hat{E}_j,U_o,W)$ i.e., $m^{VL} = m_1^{VL} \oplus m_2^{VL} \oplus \ldots \oplus m_n^{VL}$ where $n=|EXP(e_i)|$. The BPA functions m_i^{VL} are defined in Definition 3. ❑

According to Definition 2, an event is considered to have an explanation if at least one of the alternative explanations found for it is valid (see disjunction $\vee_{\hat{E}_j \in EXP(e_i)} VL(e_i,\hat{E}_j,U_o,W)$), and not to have an explanation if none of its alternative explanations is valid (see conjunction $\wedge_{\hat{E}_j \in EXP(e_i)}(\neg VL(e_i,\hat{E}_j,U_o,W))$. The functional form of m^{VL} is given by Theorem 2.

Theorem 2: *Let e_u be a runtime event, $EXP(e_u)$ be a non empty set of explanations of e_u, and $VL(e_u,\hat{E},U_o,W)$ and $\neg VL(e_u,\hat{E},U_o,W)$ be the propositions defined in Definition 2.*

(a) The combined basic probability assignment m^{VL} defined as $m^{VL} = m_1^{VL} \oplus m_2^{VL} \oplus \ldots \oplus m_n^{VL}$ will be the function:

$$m^{VL}(x) = \begin{cases} \Pi_{i \in I} m_i^{VL}(VL(e_u,\hat{E}_j,U_o,W)) \times \Pi_{j \in I'} m_j^{VL}(\neg VL(e_u,\hat{E}_j,U_o,W)) & \text{if } x = \cap_{i \in I} VL(e_u,\hat{E}_j,U_o,W) \cap_{j \in I'} \neg VL(e_u,\hat{E}_j,U_o,W) \\ 0 & \text{if } x \neq \cap_{i \in I} VL(e_u,\hat{E}_j,U_o,W) \cap_{j \in I'} \neg VL(e_u,\hat{E}_j,U_o,W) \end{cases}$$

(b) The beliefs to the propositions $\vee_{\hat{E}_j \in EXP(e_u)} VL(e_u,\hat{E}_j,U_o,W)$ and $\wedge_{\hat{E}_j \in EXP(e_u)} \neg VL(e_u,\hat{E}_j,U_o,W)$ produced by the belief function induced by m^{VL} will be:

$Bel^{VL}(\vee_{\hat{E}_j \in EXP(e_u)} VL(e_u,\hat{E}_j,U_o,W)) = \Sigma_{I \subseteq EXP(e_u) \,\&\, I \neq \emptyset}(-1)^{|J|+1}\{\Pi_{\hat{E}_j \in I} m_j^{VL}(VL(e_u,\hat{E}_j,U_o,W))\}$

$Bel^{VL}(\wedge_{\hat{E}_j \in EXP(e_u)} \neg VL(e_u,\hat{E}_j,U_o,W)) = \Pi_{\hat{E}_j \in EXP(e_u)} m_j^{VL}(\neg VL(e_u,\hat{E}_j,U_o,W))\}$

where m_j^{VL} (j=1,...,|EXP(e_u)|) are the basic probability assignment functions defined in Definition 3.

The proof of Theorem 3 is given in [38]. Based on this theorem, the belief in the existence or not of a valid explanation for a runtime event, if it is possible to identify at least one explanation through the abductive reasoning process described in Sect. IV, is computed as:

$$\mathrm{Bel}^{VL}(P) = \begin{cases} \Sigma_{J\subseteq EXP(e_i)\ \&\ J\neq\emptyset} (-1)^{|J|+1} \{\Pi_{\hat{E}_j\in J} m_j^{VL}(VL(e_i,\hat{E}_j,U_o,W))\} & \text{if } P = E^{e_i,U_o,W} \\ \Pi_{\hat{E}_j\in EXP(e_i)} m_j^{VL}(\neg VL(e_i,\hat{E}_j,U_o,W)) & \text{if } P = \neg E^{e_i,U_o,W} \end{cases}$$

In cases when $EXP(e_i)=\emptyset$, $\mathrm{Bel}^{VL}(E^{e_i,U_o,W}) = \alpha_1$ and $\mathrm{Bel}^{VL}(\neg E^{e_i,U_o,W}) = 1 - \alpha_1$. These formulas result from (a5) and the definition of m^{VL}.

D. Belief in explanation validity

The BPA to the validity of an explanation is defined as:

Definition 3: The BPA for the validity of an explanation $\hat{E}_j$ for a runtime event e_i is defined as:

(a) if $GN(e_i, \hat{E}_j U_o, W) \neq \emptyset$

$$m_j^{VL}(P) = \begin{cases} m^{CN}(\bigvee_{e_j\in CN(e_i,\hat{E}j,U_o,W)} G^{e_j,U_o,W}) & \textit{if } P = VL(e_i,\hat{E}_j,U_o,W) \\ m^{CN}(\bigwedge_{e_j\in CN(e_i,\ \hat{E}j,U_o,W)} \neg G^{e_j,\ U_o,W}) & \textit{if } P = \neg VL(e_i,\hat{E}_j,U_o,W) \end{cases}$$

(b) if $GN(e_i, \hat{E}_j,U_o,W) = \emptyset$

$$m_j^{VL}(P) = \begin{cases} \alpha_2 & \textit{if } P = VL(e_i,\hat{E}_j,U_o,W) \\ 1 - \alpha_2 & \textit{if } P = \neg VL(e_i,\hat{E}_j,U_o,W) \end{cases}$$

where,

- *$GN(e_i,\hat{E}_j,U_o,W)$* is the set of the consequent events that the explanation $\hat{E}_j$ has in the time range defined by W excluding any events in U_o and e_i or, formally, *$GN(e_i,\hat{E}_j,U_o,W) = \{c \mid (\{\hat{E}_j, AS, L\} \models c)$ and $(c\notin U_o)$ and $(c\neq e_i)$ and $(t_{mid} - w/2 \leq t_c^L)$ and $(t_c^U \leq t_{mid}+w/2)\}$* where , AS is the set of system assumptions and L is the monitor's event log.
- α_2 is a parameter taking values in [0,1] that provides the default basic probability of the validity of an explanation that has no consequences (α_2 should be set to 0 if such explanations should be totally disregarded for a given system).
- m^{CN} is the belief function resulting from combining the BPAs to the genuineness of the consequences of individual explanations, i.e., $m^{CN} = m_1^{GN} \oplus m_2^{GN} \oplus ... \oplus m_n^{GN}$ where $n=|GN(e_i,\hat{E},U_o,W)|$. ❑

m^{VL} computes the basic probability of the validity of an explanation $\hat{E}$ as the basic probability of at least one of the consequences of $\hat{E}$ being a genuine event, and the basic probability of the non-validity of $\hat{E}$ as the basic probability of none of the consequences of $\hat{E}$ being a genuine event.

The functional form of m^{ECN} is given by the following theorem.

Theorem 3: *Let $\hat{E}$ be a possible explanation for a runtime event e_u, $GN(e_u,\hat{E},U_o,W)$ be the set of the consequences that $\hat{E}$ has in the time range defined by W excluding events in U_o and e_u, and $G^{e,U_o,W}$, $\neg G^{e,U_o,W}$ be the propositions denoting the genuineness and non genuineness of the events e $GN(e_u,\hat{E},U_o,W)$, respectively.*

(a) The combined basic probability assignment m^{CN} defined as $m^{CN} = m_1^{GN} \oplus m_2^{GN} \oplus ... \oplus m_n^{GN}$ will be the function:

$$m^{CN}(x) = \begin{cases} \Pi_{i\in I} m_i^{GN}(G^{e_i,\ U_o,W}) \times \Pi_{j\in I'} m_j^{GN}(\neg G^{e_j,\ U_o,W}) & \textit{if } x = \cap_{i\in I} G^{e_i,U_o,W} \cap_{j\in I'} \neg G^{e_j,U_o,W} \\ 0 & \textit{if } x \neq \cap_{i\in I} G^{e_i,U_o,W} \cap_{j\in I'} \neg G^{e_j\ U_o,W} \end{cases}$$

where m_j^{GN} (j=1,...,|GN(e_u,$\hat{E}$,U_o,W)|) are the basic probability assignment functions defined in Definition 1.

(b) The beliefs to the propositions $\bigvee_{e_j\in CN(e_u,\hat{E},U_o,W)} G^{e_j,U_o,W}$ and $\bigwedge_{e_j\in CN(e_u,\hat{E},U_o,W)} \neg G^{e_j,U_o,W}$ produced by the belief function induced by m^{CN} will be:

$$Bel^{CN}(\bigvee_{e_j\in CN(e_u,\hat{E},U_o,W)} G^{e_j,U_o,W}) = \Sigma_{J\subseteq CN(e_u,\hat{E},U_o,W) \text{ and } J\neq\emptyset} (-1)^{|J|+1} \{\Pi_{e_j\in J} m_j^{GN}(G^{e_j,U_o,W})\}$$

$$Bel^{CN}(\bigwedge_{e_j\ \in\ CN(e_u,\hat{E},U_o,W)} \neg G^{e_j,\ U_o,W}) = \Pi_{e_j\ \in\ CN(e_u,\hat{E},U_o,W)} m_j^{GN}(\neg G^{e_j,U_o,W})\}$$

The proof of Theorem 4 is given in [38]. Based on this theorem, the belief in the validity of an individual explanation that has a non empty set of consequences is computed as:

$$Bel^{CN}(P) = \begin{cases} \Sigma_{J \subseteq CN(e_i,\hat{E},U_o,W)\ \&\ J \neq \emptyset} (-1)^{|J|+1} \{\Pi_{e_j \in J}\ m_j^{GN}(G^{ej,U_o,W})\} & \text{if } P = VL(e_i,\hat{E},U_o,W) \\ \Pi_{e_j \in CN(e_i,\hat{E},U_o,W)} m_j^{GN}(\neg G^{ej,U_o,W}) & \text{if } P = \neg VL(e_i,\hat{E},U_o,W) \end{cases}$$

In cases where an explanation $\hat{E}$ for an event has no consequences other than the event it was created to explain itself, the belief in the validity of the explanation is computed as:

$$Bel^{CN}(P) = \begin{cases} \alpha_2 & \text{if } P = VL(e_i,\hat{E},U_o,W) \\ 1 - \alpha_2 & \text{if } P = \neg VL(e_i,\hat{E},U_o,W) \end{cases}$$

The above form of Bel^{CN} is the direct consequence of axiom (a5) and the definition of m^{CN} (Definition 3).

It should also be noted that, as proved in [38], the functions m^G, m^{VL} and m^{EX} satisfy the axioms of basic probability assignments in the DS theory. In the following, we give an example of computing event genuineness beliefs using the above belief functions.

E. Example

Fig. 5 below shows a graph representing part of the diagnosis model of the LBACS system. The graph has been extracted from diagnostic assumptions for LBACS, expressed in EC-Assertion. The nodes and edges of the graph represent EC-Assertion formulas as indicated in the key of the figure.

Based on this model, the *Explain* algorithm of Fig. 3 would generate the following explanation for an event *E:Happens(e(e1,s1,r1,login(u1,101,n1),c1),10050,R(10050)* in the monitor's log:

Ê: Happens(e(e2, …, InPremises(101,n1),c2),t2, R(9050,10050))

Ê would be generated by abduction from the assumption A1 in the graph of Fig. 5. From *Ê*, the following expected consequences would also be derived by deduction:

- *C1:Happens(e(_x,…,signal(101),…),t_1,R(6050,10050))* (C1 is derived by deduction from the assumption corresponding to the edge *InPremises* → *signal* in Fig 5)
- *C2:Happens(e(_x,…,login(_U,101,_NS),…),t_1,R(8050,10050))*
  (C2 is derived by deduction from the assumption corresponding to the edge *InPremises* → *login*)
- *C3:Happens(e(_y,…,accessTo(101),…),t_2,R(9050,69050))*
  (C3 is derived by deduction from the assumption corresponding to the edge *InPremises* → *accessTo*)

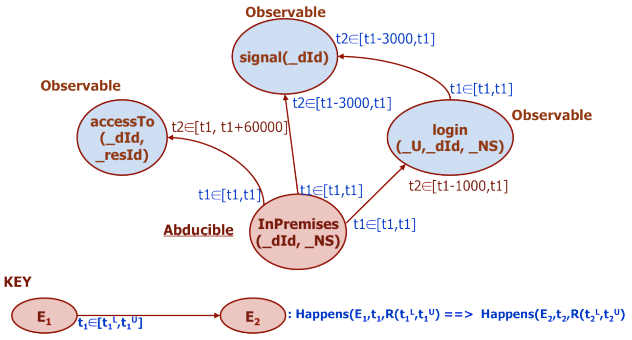

Figure 5. Part of LBACS diagnosis model

TABLE II. EXPLANATION VALIDITY AND EVENT GENUINENESS BELIEFS

Event Log	$m^{VL}(\hat{E})$	$m^G(E)$
Happens(e(…,signal(101)…), 8050, R(8050,8050)	$\alpha_2 + \alpha_2 - \alpha_2 \times \alpha_2 = 0.36$	0.36
Happens(e(…,signal(101)…), 8050, R(8050,8050); Happens(e(…,accessTo(101)…), 9801, R(9801, 9801)	$\alpha_2 + (\alpha_2 + \alpha_2 - \alpha_2 \times \alpha_2) - \alpha_2 \times (\alpha_2 + \alpha_2 - \alpha_2 \times \alpha_2) = 0.488$	0.488

Thus, depending on the log of events, the beliefs in the validity of *Ê* and the genuineness of E would be as shown in Table II (the computations in Table II assume that $\alpha_2 = 0.2$ and W = 100000 milliseconds).

The first row in the table shows the case where only the first of the consequences of *Ê* (C1) matches with an event in the log. The second row shows the case where two of the consequences of *Ê* (C1 and C3) match

with events in the log. The belief in the validity of $\hat{E}$ increases in the second case, as there are more events in the log confirming the consequences of $\hat{E}$.

V. EVALUATION

A. Objectives and experimental set up

To evaluate our approach, we have performed a series of experiments aimed at investigating: (a) the accuracy of the event assessment process, (b) possible factors that may affect this accuracy, and (c) the time required for the execution of the assessment process.

The experimental evaluation was based on a simulation of the LBACS industrial case study. The use of simulation was necessary in order to be able to introduce attacks in the communication lines of the system and investigate their effect on the assessment process. The model of LBACS used in the simulation is shown in Fig. 6. This model covered all the components of LBACS and associated the communication lines between these components with *adversaries* able to intercept, and block or delay events transmitted over the lines.

The simulation model was used to generate different event logs for analysis. Each event log was generated by producing random *seed events* triggering interactions between LBACS components and then using the model to produce further events based on the specification of the components of the model. A request for accessing an LBACS resource sent by a mobile device to the access control server, for example, was a seed event generated at a random time point and with event parameter values (e.g., device ID) that were randomly selected from pre-specified sets of values having an equal probability of selection. Upon the receipt of such a seed request, the access control server would request the location of the device from the location server. The latter server would then respond according to its specification and with a random delay. Furthermore, adversaries could intervene in interactions by dropping or delaying randomly selected events. *Adversary03* in Fig. 6, for example, could delay the request to the access control server or drop it altogether thus stopping the event sequence. Following a random choice to delay an event, an adversary introduced a random delay to it.

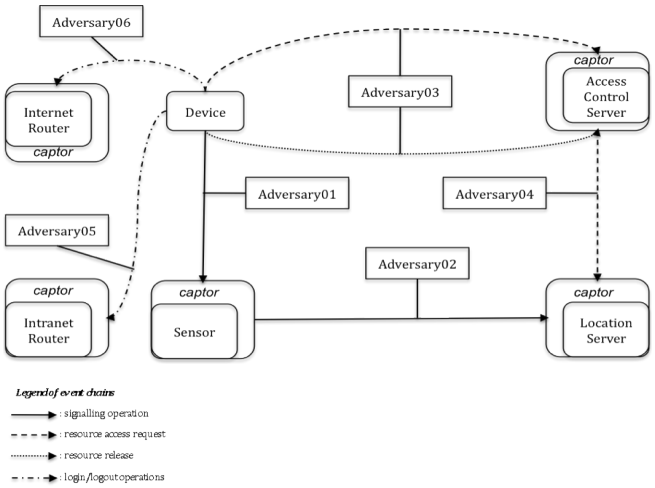

Figure 6. LBACS simulation model

Based on this process, we generated three groups of simulated events (G1, G2, and G3) to represent different levels of adversary intervention. Adversary intervention levels, or "attack sizes", were measured by the number of the events affected by adversaries in a simulation (i.e., the number of delayed or dropped events). The level of intervention was set to 10 percent for G1, 20 percent for G2 and 30 percent for G3. Each of G1, G2 and G3 consisted of five different events sets consisting of 5000 random events each. All the five sets in each group had the same level of adversary intervention. The generated event sets were subsequently fed into EVEREST to execute monitoring and event assessment. In this process, we used 18 diagnostic assumptions.

The accuracy and degree of completeness of the event genuineness assessment process was measured separately for genuine and fake events, using the following formulas:

$$P_f = |Events_{BR} \cap Events_f| / |Events_{BR}| \quad (1)$$

$$P_g = |Events_{BR} \cap Events_g| / |Events_{BR}| \quad (2)$$

$$R_f = |Events_{BR} \cap Events_f| / |Events_f| \quad (3)$$

$$R_g = |Events_{BR} \cap Events_g| / |Events_g| \quad (4)$$

In these formulas, $Events_{BR}$ is the set of events for which the assessment process generated a genuineness belief in the range *BR*, $Events_g$ and $Events_f$ are the sets of genuine events and fake events generated in a simulation respectively, and |•| is the cardinality of a set.

The first two of the above formulas measure the precision (accuracy) of the assessment process. More specifically, (1) measures the ratio of events with a genuineness belief in the range BR that are fake, and (2) measures the ratio of events with a belief in the range BR that are genuine (note that $P_g = 1–P_f$). The next two formulas (i.e., (3) and (4)) measure the recall (i.e., extent of completeness) of the assessment process. In particular, (3) measures the ratio of fake events whose belief was in the range BR and (4) measures the ratio of genuine events whose belief was in the range BR.

In addition to different *types of events (i.e., fake* and *genuine)* and different *ranges of genuineness beliefs*, the precision and recall of the assessment process was investigated with respect to different sizes of *assessment windows* (W), and different *attack sizes* (AS). In particular, we investigated accuracy for:

- Three different belief ranges, namely a *low belief range* set to [0, 0.3), a *medium belief range* set to [0.3, 0.7) and a *high belief range* set to [0.7, 1.0].
- Five different assessment windows (i.e., windows of 1.5, 2.5, 5, 7.5 and 10 seconds).
- Three different attack sizes (i.e., simulations where the events affected by adversaries accounted for 10%, 20% and 30% of the entire event log produced in the simulation).

The outcomes of this investigation are discussed below.

B. Precision and recall of assessment

Tables III and IV show the precision and recall for genuine and fake events in the experiments, respectively. Precision and recall are shown for different belief ranges (BR) and sizes of the assessment window (W). The precision (P_g and P_f) and recall measures (R_g and R_g) shown in the tables have been computed as averages across the five different sets of events that were generated in the simulations (i.e., a total of 25,000 simulated events), assuming an attack size of 20% (i.e., simulations where one in every five events was fake). The tables show also the average recall and precision measures, as computed across all five assessment windows, for the different belief ranges and genuine/fake events (see columns AVE_W in the tables). Graphs of the AVE_W measures are also shown in Figures 7 and 8.

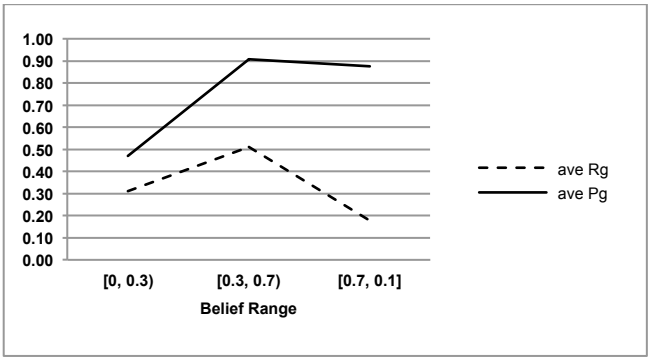

Figure 7. Average precision/recall of genuine events

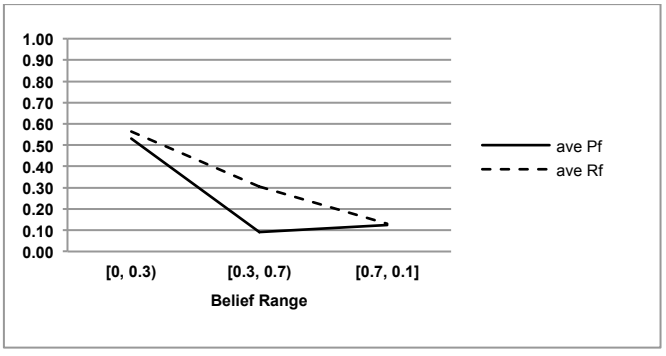

Figure 8. Average precision/recall of fake events

As Figure 7 shows, the average precision for genuine events (P_g) grows, as expected, from 0.47 at the lower belief range to 0.88 at higher belief range. The same pattern was mirrored in the case of precision of

fake events (P_f) but in the opposite direction (i.e., P_f decreased from the lower to the higher belief range as $P_f = 1 - P_g$) as indicated in Figure 8[4].

The average recall for genuine events (R_g) peaked at 0.51 in the middle belief range (i.e., 51% of genuine events had genuineness belief measures in that range) and showed unexpectedly high and low values for the lower and higher belief range (i.e., 0.31 and 0.18, respectively). The average recall measures for fake events (R_f) were in line with expectations dropping from 0.57 at the lower belief range down to 0.13 at the higher belief range.

Overall, the experimental results indicated that genuineness belief measures were capable of spotting with accuracy genuine and fake events. In particular, belief measures higher than 0.3 were found to be a good indicator of event genuineness and belief measures below 0.3 were found to be a good indicator of fake events. The experiments also indicated that belief measures shown a reasonable recall performance in the case of fake events but mixed performance in the case of genuine events.

TABLE III. PRECISION/RECALL OF GENUINE EVENTS W.R.T TO ASSESSMENT WINDOW AND BELIEF RANGE

		Assessment Window (W)					
	Belief Range (BR)	**1.5**	**2.5**	**5**	**7.5**	**10**	**AVE_W**
R_g	**[0.0, 0.3)**	0.00	0.01	0.37	0.41	0.77	0.31
	[0.3, 0.7)	0.53	0.63	0.59	0.58	0.22	0.51
	[0.7, 1.0]	0.47	0.36	0.05	0.01	0.01	0.18
P_g	**[0.0, 0.3)**	0.00	0.13	0.74	0.70	0.78	0.47
	[0.3, 0.7)	0.84	0.86	0.90	0.94	1.00	0.91
	[0.7, 1.0]	0.85	0.90	0.63	1.00	1.00	0.88

TABLE IV. PRECISION/RECALL OF FAKE EVENTS W.R.T TO ASSESSMENT WINDOW AND BELIEF RANGE

		Assessment Window (W)					
	Belief Range (BR)	**1.5**	**2.5**	**5**	**7.5**	**10**	**AVE_W**
R_f	**[0.0, 0.3)**	0.13	0.29	0.58	0.82	1.00	0.57
	[0.3, 0.7)	0.47	0.50	0.37	0.18	0.00	0.30
	[0.7, 1.0]	0.40	0.21	0.05	0.00	0.00	0.13
P_f	**[0.0, 0.3)**	1.00	0.87	0.26	0.30	0.22	0.53
	[0.3, 0.7)	0.16	0.14	0.10	0.06	0.00	0.09
	[0.7, 1.0]	0.15	0.10	0.37	0.00	0.00	0.12

The more detailed figures in Tables III and IV show also some performance differences across the different assessment windows. Most notably, increasing the assessment window can lead to increased precision in the high belief range. More specifically, as Table III indicates, P_g increased from 0.85 to 1.00 when W increased from 1.5 secs to 10 secs (correspondingly P_f decreased from 0.15 to 0 across these W sizes, as indicated in Table IV). Increasing the assessment window, however, made low beliefs less accurate indicators of fake events. In particular, for beliefs in the range [0.0, 0.3), P_f fell from 1.0 to 0.22 and, correspondingly, P_g increased from 0.29 to 0.76.

Overall, the most accurate assessment of fake and genuine events was observed for the assessment windows of 1.5 and 2.5 seconds. For these assessment windows the aggregate probability of making a wrong assessment, i.e., characterizing a fake event as genuine based on a belief higher than 0.7 or a genuine event as fake based on a belief less than 0.3 was 0.15 and 0.26, respectively (this probability is computed as the sum of $P_{f\,[0.7,1.0]} + P_{g[0.0,0.3)}$). The most likely explanation of the better performance in the case of the two smaller assessment windows is that length of these windows was closer to the average time difference between the events used in the monitoring rules and assumptions of the experiments. The recall (R_g) of genuine events in the low belief range [0, 0.3) was, as expected, very low (almost 0 per cent) for the two shorter assessment windows (W=1.5 and W=2.5). This performance deteriorated as W increased.

To assess the statistical significance of the observed differences in the R_g, R_f, P_g, and P_f measures across different belief ranges and sizes of assessment windows, we performed two-way analysis of variance [14].

This analysis shown that the differences observed in P_g and P_f measures across different belief ranges were statistically significant in at α=0.05 (P-value: 0.022, F-value: 6.344). Unlike them, the differences observed for R_g and R_f across different belief ranges were not found to be statistically significant at α=0.05.

[4] The P_f line shown in Figure 8 is symmetrically opposite to the P_g line in Figure 7. The reason for showing it, in addition to the P_f line is to enable a direct visual comparison with R_f in a single graph.

The analysis of variance indicated also that the differences observed in precision and recall measures across different assessment windows were not statistically significant at α=0.05.

In the experiments, we also investigated the accuracy of assessment for different attack sizes. Tables V and VI show recall and precision measures for genuine and fake events, respectively, for different belief ranges, and attack sizes of 10%, 20% and 30%. These measures were computed as averages over five different random event sets for each attack size (i.e., a total of 15,000 events) whilst W equaled 2.5secs in all experiments.

As shown in the tables, the precision of genuine events (P_g) with a belief in the high belief range [0.7, 1.0] dropped from 0.94 to 0.77 as the attack size increased from 10% to 30%. Similarly the precision of fake events (P_f) with a genuineness belief in the low belief range [0.0, 0.3) dropped as the attack size increased (P_f was 1.0 for AS=10%, 0.95 for AS=20% and 0.98 for AS=30%). The analysis of variance of P_g and P_f indicated that the differences observed in these measures across different attack sizes were not of statistical significance at α=0.05.

TABLE V. PRECISION AND RECALL FOR GENUINE EVENTS W.R.T TO ATTACK SIZE AND BELIEF RANGE

		Attack size (AS)		
	Belief Range (BR)	**10%**	**20%**	**30%**
R_g	**[0.0, 0.3)**	0.00	0.00	0.00
	[0.3, 0.7)	0.66	0.60	0.61
	[0.7, 1.0]	0.34	0.40	0.38
P_g	**[0.0, 0.3)**	0.00	0.05	0.02
	[0.3, 0.7)	0.94	0.85	0.76
	[0.7, 1.0]	0.94	0.90	0.77

TABLE VI. PRECISION W.R.T TO ATTACK SIZE AND BELIEF RANGE

		Attack size (AS)		
	Belief Range (BR)	**10%**	**20%**	**30%**
R_f	**[0.0, 0.3)**	0.22	0.20	0.26
	[0.3, 0.7)	0.37	0.56	0.50
	[0.7, 1.0]	0.40	0.24	0.24
P_f	**[0.0, 0.3)**	1.00	0.95	0.98
	[0.3, 0.7)	0.06	0.15	0.24
	[0.7, 1.0]	0.06	0.10	0.23

Recall measures showed a mixed performance across different attack sizes that cannot lead to some conclusion about the effect of the assessment window size upon it. This was confirmed by the analysis of variance of R_g and R_f which indicated that the differences of R_g and R_f measures across different attack sizes were not of statistical significance at α=0.05.

C. Belief computation time

Fig. 9 shows the average time required for the computation of an event genuineness belief measure for assessment windows (W) of different size. The average time shown was calculated across a total of 25,000 computations of belief measures for each window size.

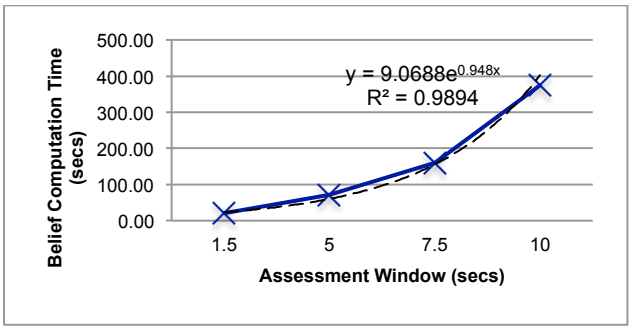

Figure 9. Average computation time for genuineness beliefs

As shown in the figure, the belief computation time increases exponentially along with increments in W since, as W becomes larger, there are more runtime events to consider whilst searching for matches to the

expected consequences of the alternative explanations of each event whose genuineness is assessed. Fig. 9 shows also the exponential trend line that was fitted to the data using regression. The correlation co-efficient of this line (i.e., R^2=0.9894) was found to be statistically significant at *a=0.05,* confirming the effect of the assessment window onto the computation time of genuineness beliefs.

The exponential increase of the belief computation time along with W increments raises a question about the scalability of our approach when applied to complex systems. In response to this question, we should note that there are ways of controlling the exponential increase of the computational cost of the assessment process. One such way is to keep the size of W small to ensure a timely generation of the required beliefs. Our initial experiments have indicated that this would be possible without affecting the accuracy of the assessment, as demonstrated by the experimental data in Table III.

The computational cost of our approach depends also on the number of the diagnostic assumptions that can be applied for generating an explanation: it can grow exponentially as the number of assumptions increases. However, in many cases, this cost could be controlled through the adequate (re-)formulation of the diagnostic model. More specifically, it is possible to omit some intermediate diagnostic assumptions and retain in a diagnostic model only those, which connect directly a final event outcome with the event(s) that are the root-cause of it. To appreciate how consider, for example, a set of *n-1* diagnostic assumptions producing an explanation path $e_1 \rightarrow e_2 \rightarrow ... \rightarrow e_n$ where

- e_n is an observable event;
- e_1 is an observable event or an internal non observable system event that can cause an observable outcome other than e_n; and
- $e_2, ..., e_{n-1}$ are derived events that cannot cause any observable outcomes other than e_n.

In this example, it would be possible to reduce the diagnostic model to one assumption generating directly the path $e_1 \rightarrow e_n$ and save time during the search for confirmed explanations for e_n. In approaches, which use abductive reasoning to provide a full diagnostic explanation of the trace of events within a system that has led to a specific outcome, the reduction of a diagnostic assumptions set, as the above example, would not be possible without affecting the quality of diagnosis. In our approach, however, this is not a problem as in the chain $e_1 \rightarrow e_2 \rightarrow ... \rightarrow e_n$ the only events that can affect the computation of belief measures are e_1 and e_n. Note also that the diagnostic model for a system can also be reduced through static analysis. More specifically, it is possible to use static analysis to identify those assumptions in the model that involve only derived events in their head and only non observable events in their body, which furthermore cannot directly cause any observable events other than the event e_n of interest. If such diagnostic assumptions exist, they can be removed from the diagnostic model, and be replaced by alternative assumptions expressing the relations between the causes of the events in their body and the effects of the events in their head.

Following the trimming of a diagnosis model, a further optimisation would be to pre-compute symbolically and maintain a graph of: (a) all the possible abstract (i.e., non-grounded) explanations for each observable event in the model, and (b) all the possible abstract consequences of these explanations. More specifically, for each observable event that is represented by a *Happens* predicate in the head of a diagnostic assumption in the model, it is possible to compute an explanation tree and, for each of the leaf events in this tree, to compute its possible consequences tree. Subsequently, when a genuineness assessment is required for a concrete event matching the original *Happens* predicate at runtime, the generated explanation and consequences trees could be instantiated directly to generate the explanations for the event of interest and their potential confirmation events. This would make it unnecessary to search for explanations and deduce consequences from scratch at runtime by using the diagnostic assumptions of the model.

D. Discussion

The undertaken experimental evaluation of our approach has focused on exploring key factors that may affect its performance. Whilst the initial results are positive, especially with regards to the accuracy (precision) of the assessment, further investigation is required in order to investigate the recall performance of our approach, and to explore additional factors that may affect other aspects of its performance.

In particular, it will be interesting to explore the sensitivity of the accuracy of event genuineness assessments when using diagnostic models (i.e., sets of diagnostic assumptions) of varying degrees of completeness with respect to the behavior of a system under surveillance. It will also be interesting to explore the sensitivity of the accuracy of genuineness assessments when using diagnostic models of varying degrees of correctness (e.g., assumptions specifying incorrect time intervals between the events that they involve). Depending on the assessment window that is used, it should be noted that the latter factor might affect not only the accuracy but also the efficiency of our approach.

It will be also interesting to explore the effect of adversaries with more advanced capabilities, including the ability to introduce groups of interrelated fake events into a system in order to overcome the basic premise of the diagnosis model, i.e., the principle of seeking confirmation of an event through other events that can be the outcome of the same cause. In the LBACS case study, for instance, an attacker with such

capabilities should be able to generate multiple fake signals for a given mobile device and fake resource access requests. This would compromise the ability of our approach to detect the fake events, if the given type of fake events is not related to several other event types through common causes in the underlying diagnosis theory.

It should be noted that whilst attackers with such capabilities would be able to overcome the detection power of our approach, in reality to be effective, such attackers should have or be able to develop knowledge of the exact diagnosis theory that it deploys for detecting fake events for a given system. More specifically, to be effective an attacker should have or be able to derive knowledge about co-occurrences of events in diagnostic assumptions, events with potentially common causes, time dependencies between events in explanation chains, and time dependencies between events with common causes. This, in our view, would be difficult for complex diagnostic theories.

Further experimentation will also be needed in order to investigate the sensitivity of our approach to parameters *a1* and *a2*, and especially when varying these values along with the total number of assumptions in a diagnostic theory. It will also be necessary to explore whether there is some systematic drop in the accuracy of the assessment for particular ranges of assessment window sizes (as in the case of W=5 in our simulations which led to a drop in accuracy that seemed to be an outlier point).

Lastly, we should note that an experimentation based on some benchmark data set would be useful. This has not been possible so far, as existing benchmarks for security or dependability (e.g. [43]) do not provide detailed specifications of the internal behaviour of their underlying systems that would enable us construct a diagnosis theory to use with our approach.

VI. RELATED WORK

The problem of assessing the trustworthiness of runtime events used in runtime software system monitoring is relevant to work on runtime system verification, and diagnosis in the fields of software engineering and systems security. It is also relevant to work on abductive reasoning and diagnosis in AI. In the following, we present an overview of relevant strands of work in these areas, and position our approach within it.

Runtime verification has been the focus of several approaches and systems (e.g., [5][6][7][10][12][27][42]). The focus of runtime verification is to check whether the execution trace(s) of a software system satisfy or not a given property [25]. This check has often to rely on partial traces of system executions and may involve prediction of the remaining trace. RV systems may be distinguished by the language that they use to express properties (e.g. rule based [5][6], query based [27], or assertion based and temporal logic based approaches [4][24]), and the way in which they realise their checks (e.g. *intrusive approaches* where monitoring code is weaved into the code of the system to be monitored [10][5] vs. *non intrusive approaches* that deploy external monitors [35][36]).

Examples of general-purpose RV systems include Java-MoP [28], JPaX [17], Java-MaC [19], Jassda [9], Temporal-Rover [13], JNuke [4] and JPF [52]. Java-MoP [28] realizes the monitoring oriented programming (MoP) approach in which monitoring code is synthesized automatically from assertions specified by programmers and is weaved into system's code to perform the runtime checks. JPaX (Java PAth eXplorer [17]) carries out various forms of error pattern analysis to detect potential violations of properties expressed as past and future LTL formulas (e.g. deadlocks) by instrumenting Java byte code. Java-MaC [19] performs runtime checks of safety properties of Java programs. These properties are expressed using a primitive and a meta-event definition language (PEDL and MEDL), enabling the definition of code level events (PEDL) and high-level events (MEDL) defined in terms of low-level events to enable the specification of the required properties. Java-MaC generates monitors automatically. Jassda [9] checks assertions by observing traces of Java program events generated for debuggers through the Java Debug Interface (JDI). Java-MaC assertions are written in a CSP-like language. Temporal Rover [13] is a commercial toolkit supporting the runtime verification of temporal properties over C, C++, VHDL, Verilog, and ADA programs. These properties are expressed as linear or metric temporal logic (LTL/MTL) assertions that are embedded as comments the source code of the program to be monitored. JNuke [4] is an integrated runtime verification and model-checking framework. JNuke offers a specialized VM and an RV API supporting backtracking in Java program execution and access to the program state. Backtracking is supported through the specification of program checkpoints that allow the exploration of alternative program execution paths at runtime. Custom checking algorithms can be implemented on top of JNuke. JPF (JavaPathFinder) [52] is a model checker for Java byte code offering a specialized Java Virtual Machine (JVMJPF), supporting exhaustive program execution. JPF searches the state space of the checked program for "low-level" properties like deadlocks, unhandled exceptions and/or failed assertions and is extensible via a listener API to external monitors wishing to check more general properties.

There are also systems focusing on runtime verification of security properties. Naldurg et al [29] have developed a framework for intrusion detection using temporal patterns, specified in EAGLE [6]. These

patterns enable reasoning about the data values observed in individual events. Thus, their framework supports the monitoring of attacks whose signatures have statistical properties. Ko et al. [21] have also used RV techniques to detect vulnerabilities of security-critical programs. In their framework, trace policies, expressed in "parallel environment grammar (PE-grammar)" determine security-valid operation sequences monitored against program executions. PE-grammar can express various classes of security trace policies, including access to system objects, synchronization, operation sequencing, and race conditions in concurrent or distributed programs. Execution Monitoring [34] is an approach for monitoring violations of security policies, based on the security automata (i.e., automata with control capabilities) that can terminate system execution if would violate a security policy. Finally, EVEREST, the monitoring framework integrated with the event assessment approach described in this paper, has been developed to support the monitoring of security and dependability properties of distributed software systems and/or software systems that may integrate components dynamically [35].

Diagnosis of software system faults and violations of desired system properties has also been the focus of several strands of works [8][15][31][37], which view it as the identification of trajectories of events that have caused the faults or violations of the desired properties. Typically, the different approaches to diagnosis use automata expressing the expected behaviour of the monitored system and try to synchronize them with the event traces generated by the system under surveillance for carrying out the diagnostic task. The work in [31], for instance, adopts a decentralised diagnosis approach where automata are synchronised with individual system components (to enable fault detection at system components level), and are then aggregated for the global system to enable fault detection at system level. The work in [37] and [8] focuses on generating algorithms acting as detectors of internal system faults (aka diagnosers).

In our approach, we distinguish between monitoring and diagnosis treating the former as the task of detecting violations of system properties and the latter as the task of assessing the genuineness of events that underpin monitoring. Hence, both our focus and our approach are distinct from runtime verification and diagnosis as they appear in the literature. Our approach, however, is complementary to runtime verification and diagnostic systems as it can be used to assess the trustworthiness of the events they analyse. This capability is useful in cases of RV systems supporting non-intrusive external monitoring based on events that are produced and captured by different distributed system components.

Forms of diagnosis more similar to our work have been the focus of work in AI. Console et al. [11], for example, have proposed a temporal abduction approach, which makes use of an underlying domain theory to generate explanations for observations. Our approach differs from in two main respects: (a) it assesses the validity of explanations by evaluating the genuineness of their consequences, and (a) it treats the time constraint satisfaction problem as a linear programming problem. The approach in [32] also uses causal and temporal dependencies between events/propositions. Time in this approach is represented using Allen's interval algebra [2], and uncertainty is expressed by Bayesian probabilities assigned to dependencies. Our approach differs from [32], as we use a more elaborate representation of time and treat uncertainty through DS theory beliefs. Thus, we avoid the need to rely on a-priori probabilities of causal/temporal event dependencies, and can account for uncertainties regarding the occurrence of events. Finally, the AR system SCIFF [1] also uses the notion of confirmation of abduced information against dynamic observations (i.e., the equivalent of events in our approach). SCIFF, however, assumes that dynamic observations, which have not happened within their expected time range, disconfirm abduced information automatically. Furthermore, SCIFF does not quantify confirmation through the use of uncertainty measures (e.g. beliefs or probabilities), as ours.

VII. CONCLUSIONS

In this paper we have presented an approach for assessing the genuineness of events used for runtime monitoring of cyber systems. Our approach is based on the generation of possible explanations for runtime events using abductive reasoning and assessing whether any further *effects* that these explanations would have can be confirmed by other runtime events in the monitor's log. The confirmation of such effects provides evidence in support of the validity of explanations and, therefore, evidence that the original events are genuine. The assessment of the genuineness of events (including those generated by the system under surveillance and the explanation effects) and the validity of explanations is based on the computation of beliefs using functions grounded in the DS theory of evidence.

An initial experimental evaluation of our approach has shown promising results. Notably, high and low event genuineness beliefs have been shown to correlate with genuine and fake events. The accuracy of assessments based on such beliefs depends on the extent of the attacks that a system has been subjected to. This ability to distinguish genuine from fake events is a prerequisite for effective monitoring of security of cyber systems without a detrimental effect to their operational continuity.

Currently, our work focuses on investigating ways for analysing statically the effectiveness of the diagnostic model used for a system and updating it based on monitoring results. Our overall objective is to develop support for optimising the sets of diagnostic assumptions used for a system. In particular, we are investigating the use of static analysis of a diagnostic model to determine the coverage that it offers against a monitoring specification, to build symbolic explanation and consequence trees for all the observable events defined in the model, and to identify diagnostic assumptions that could be removed from the model without affecting the closure of the results of the event assessment process. These types of analysis and pre-processing of diagnosis model can help avoiding unnecessary computations during the generation of explanations as we discussed in Sect. V. We are also exploring the use of statistical data regarding the application of individual diagnostic assumptions in a diagnostic model and their effectiveness in producing accurate assessments of genuineness. Statistical analysis may, for example, indicate that a partcular diagnostic assumption produces unconfirmed consequent events because of the incorrect time range within which it expects the events to occur.

Finally, we are working on some limitations of our framework and the EVEREST monitoring system that underlies it. One such limitation is that the framework does not distinguish between events that could not possibly have an explanation (because there is no diagnostic assumption that could explain them) and events for which although it would – in principle – be possible to identify an explanation (as there are diagnostic assumptions having them as a consequence), no explanation is found due to the absence of observed or derived events matching the conditions required by the relevant assumptions. In its current form, the diagnosis framework does not distinguish between these two cases of events and assigns a fixed belief measure (i.e., the value of parameter α_1) to the genuineness of events with no explanation regardless of the exact reason why this is the case. These two cases can be distinguished by modifying the BPA function $m_i^{EX}(P)$ (e.g., to make it assign different belief measures in each of these cases).

With regards to EVEREST, we are extending it to support time-triggered event, i.e., events represent that the clock of a computational infrastructure reaches a particular time point T such as the end of the day, month etc.

VIII. REFERENCES

[1] Alberti M. et al., Abduction with Hypotheses Confirmation, *Int. Joint Conf. on Artificial Intelligence*, 19: 1545-1546, 2005

[2] Allen J. F., Maintaining Knowledge about Temporal Intervals. *Communications of the ACM* 26(1): 832-843, 1983

[3] Armenteros A. et al., Realising the Potential of SERENITY in Emerging AmI Ecosystems: Implications and Challenges, In *Security and Dependability for Ambient Intelligence*, Advances in Information Security, Springer, 2009

[4] Artho C, et al., JNuke: Efficient Dynamic Analysis for Java. *16th Int. Conf. On Computer Aided Verification*, LNCS 3114: 462–465, 2004

[5] Baresi L, and Guinea S, Towards Dynamic Monitoring of WS-BPEL Processes. *3rd Int. Conf. On Service Oriented Computing*, 2005.

[6] Barringer H., et al., Rule-Based Runtime Verification. *5th Int. Conf. on Verification, Model Checking, and Abstract Interpretation*, LNCS 2937: 44-57, 2004

[7] Barringer H., Rydeheard D., and Havelund K. Rule systems for run-time monitoring: From Eagle to RuleR. In *Runtime Verification*, pp. 111-125, Springer Berlin/Heidelberg, 2007

[8] Bouyer P., et al., Fault Diagnosis using Timed Automata. LNCS 3441:219–233, 2005

[9] Brörkens M., and Möller M., Jassda trace assertions, runtime checking the dynamic of java programs. *Int. Conf. on Testing of Communicating Systems*, Berlin, Germany, pp. 39-48, 2002

[10] Chen F., and Rosu G., Towards Monitoring-Oriented Programming: A Paradigm Combining Specification and Implementation. In *Electronic Notes in Theoretical Computer Science*, 89(2): 108-127, 2003

[11] Console L., Terenziani P., and Dupre D.T., Local reasoning and knowledge compilation for efficient temporal abduction. *IEEE Transactions on Knowledge and Data Engineering,* 14(6): 1230-1248, 2002

[12] d'Amorim M., and Havelund K., Event-based runtime verification of Java programs. *ACM SIGSOFT Software Engineering Notes*, *30*(4): 1-7, 2005

[13] Drusinsky D. The Temporal Rover and the ATG Rover. In *SPIN Model Checking and Software Verification*, LNCS 1885: 323–330, 2000

[14] Gelman A., Analysis of variance: why it is more important than ever (with discussion), *The Annals of Statistics*, 33(1): 1-53, 2005

[15] Grastien A, Cordier M, Largouët C, Incremental Diagnosis of Discrete-Event Systems. *15th Int. Workshop On Principles of Diagnosis*, 2005

[16] Heckerman, D., A tutorial on learning with Bayesian networks, *Innovations in Bayesian Methods*, 33-82, 2008

[17] Havelund K., and Roşu G., An Overview of the Runtime Verification Tool Java PathExplorer. *Formal Methods in System Design*, 24(2):189-215, 2004

[18] Janicke H, et al, Analysis and Run-time Verification of Dynamic Security Policies. *Defence Applications of Multi-Agent Systems*, 2006

[19] Kim M. et al., Java-MaC: a runtime assurance approach for Java programs, *Formal Methods in System Design*, 24(2):129-155, 2004

[20] Knight K., Unification: a multidisciplinary survey. *ACM Computing Surveys*, 21(1):93-124, 1989.

[21] Ko C., et al., Execution monitoring of security-critical programs in distributed systems: A specification-based approach., *IEEE Symp. on Security & Privacy*, 1997

[22] Kowalski B., Sergot M., A logic-based calculus of events, *Next Generation Computing*, 4(1): 67-95, 1986

[23] Lazarevic A., Kumar V., Srivastava J., Intrusion detection: a survey, In *Managing cyber-threats: issues approaches & challenges*, Springer, 19-78, 2005

[24] Lee I., et al. Runtime assurance based on formal specifications. *Departmental Papers (CIS)*, 294, 1999

[25] Leucker M, Schallhart C., A brief account of runtime verification, *Journal of Logic and Algebraic Programming*, 78(5): 293-303, 2009

[26] Ligatti J, Bauer L, and Walker D, Edit Automata: Enforcement Mechanisms for Run-time Security Policies. *International Journal of Information Security*, 4(1): 2-16, 2005

[27] Martin M., Livshits B., & Lam M. S. Finding application errors and security flaws using PQL: a program query language. *ACM SIGPLAN Notices*, 40(10): 365-383. 2005

[28] Meredith P. et al., An overview of the MOP runtime verification framework, *Int. J. of Software Tools for Technology Transfer*, 1-41, DOI: 10.1007/s10009-011-0198-6, 2011

[29] Naldurg P., Sen K., and Thati P. A temporal logic based framework for intrusion detection. *Formal Techniques for Networked and Distributed Systems–FORTE 2004*, 359-376, 2004

[30] Paul G., Approaches to Abductive Reasoning: an overview. *Artificial Intelligence Review*, 7(2):109-152, 1993

[31] Pencolé Y., and Cordier M.O, A formal framework for the decentralised diagnosis of large scale discrete event systems & its application to telecommunication networks. *Artificial Intelligence*, 164(1):121-170, 2005

[32] Santos Jr E., Unifying time and uncertainty for diagnosis. *Journal of Experimental and Theoretical Artificial Intelligence*, 8:75-94, 1996

[33] Shafer G., A Mathematical Theory of Evidence., Princeton University Press, 1975

[34] Schneider F.B., Enforceable security policies. *ACM Trans. Inf. Syst. Secur.* 3(1): 30-50, 2000

[35] Spanoudakis G., Kloukinas C., and Mahbub K., The SERENITY Runtime Monitoring Framework, In *Security and Dependability for Ambient Intelligence*, Advances in Information Security Series, Springer, pp. 213-238, 2009

[36] Spanoudakis G., and Mahbub K., Non intrusive monitoring of service based systems. *Int. Journal of Cooperative Information Systems*, 15(3):325–358, 2006

[37] Tripakis S, Fault diagnosis for timed automata. *7th Int. Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems*, LNCS 2469: 205–224, Springer, 2002

[38] Tsigkritis T., Diagnosing Runtime Violations of Security and Dependability Properties, PhD Thesis, City University, May 2011

[39] Tsigritis T, Spanoudakis G., Diagnosing Runtime Violations of Security & Dependability Properties, *20th Int. Conf. on Software Engineering and Knowledge Engineering*, July 2008

[40] Tsigritis T., Spanoudakis G., A Temporal Abductive Diagnostic Process for Runtime Properties Violation, In *ECAI 2008 Workshop on Explanation Aware Computing (Exact '08)*, 2008

[41] Tsigritis T. et al. Diagnosis and Threat detection capabilities of the SERENITY Runtime Framework, *In Security and Dependability for Ambient Intelligence*, Advances in Information Security Series, Springer, pp. 239-272, 2009

[42] Brat G., Havelund, K., Park S., & Visser W. Model checking programs. *IEEE Int. Conf. on Automated Sofware Engineering (ASE)*, pp. 3-12, 2000

[43] DBench Team, DBench Dependability Benchmarks, DBench Project Deliverables DBEV3, DBEV4 and CD1, May 2004, available from: http://webhost.laas.fr/TSF/DBench/

[44] Valdes A., and Skinner K., Adaptive Model-Based Monitoring for Cyber Attack Detection. *3rd Int. Workshop on Recent Advances in Intrusion Detection* (RAID '00), 2000

[45] Shrestha S., Balachandran M., Agarwal M., Phoha V. V., & Varahramyan K. ,A chipless RFID sensor system for cyber centric monitoring applications., *IEEE Transactions on Microwave Theory and Techniques*, *57*(5):1303-1309, 2009

[46] Pietro R., and Mancini L.V. (eds). *Intrusion Detection Systems*. Springer, 2008.

[47] Debar H., Dacier M., and Wespi A. Towards a taxonomy of intrusion-detection systems. *Computer Networks* 31(8): 805-822, 1999.

[48] Roman R., Zhou J., and Lopez J. Applying intrusion detection systems to wireless sensor networks., *IEEE Consumer Communications and Networking Conference (CCNC'06)*. 2006.

[49] Singhal A., "Intrusion Detection Systems." *Data Warehousing and Data Mining Techniques for Cyber Security* 43-57, 2007.

[50] Miller K.W., Voas J., Hurlburt G.F., BYOD: Security and Privacy Considerations, *IT Professional*, 14(5): 53-55, Sep-Oct 2012.

[51] Muzammil K., Ahmed A., and Cheema A.R., Vulnerabilities of UMTS access domain security architecture. *9th Int. Conf. on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing*, 2008. SNPD'08. 2008.

[52] Visser W, Havelund K, Brat G, and Park SJ, Model Checking Programs. 15th *IEEE Conference on Automated Software Engineering*, 2000