Collaborative Attribute Retrieval in Environment with Faulty Attribute Managers

Mario Faiella, Fabio Martinelli, Paolo Mori, Andrea Saracino, Mina Sheikhalishahi
Istituto di Informatica e Telematica
Consiglio Nazionale delle Ricerche
Pisa, Italy
Email: name.surname@iit.cnr.it

Abstract—Attributes describing the features of subjects, objects and of the environment are used in access and usage control models to determine the right of a subject to use an object in a given environment. Hence, it is crucial for the effective enforcement of access and usage policies that authorization systems are able to promptly retrieve the values of the required attributes from the Attribute Providers. However, sometimes attribute providers could not respond when queried by Authorization systems, because they could be temporarily down or unreachable. This could affect the decision processes, causing some requests to be unduly denied or some ongoing accesses to be unduly interrupted. This paper proposes a strategy that can be adopted by an Authorization system to estimate the value of the attributes it requires when the corresponding attribute providers are not responding. This strategy leverages on the collaboration of the other Authorization systems which exploit the same attribute providers, and which could have cached a value for the required attributes. We validate the presented approach through a set of simulative experiments which consider the presence of malicious authorization systems in the cooperative environment.

I. INTRODUCTION

Attribute-based access control models determine the right of a subject to access an object by checking the values of a set of attributes. Attributes are used to describe the properties of the subject, of the object, and of the execution environment involved in the access. Each attribute can take one or multiple (a bag of) values. For instance, in the administrative domain of a company, the role is an attribute of the subject which describes the role of the employees in the company. Possible values for the role attribute could be: ceo, department head, project leader, sysadmin, and developer, with the obvious meanings.

In traditional access control models, attributes are static (immutable), because their values do not change frequently over time, and they can be updated only through an explicit administrative action performed by a system administrator. With reference to the previous example, we consider the role of a subject as a static attribute because it is updated by the system administrator, for instance as a consequence of a career advancement. The Usage Control model encompasses and extends the traditional access control models. It introduces mutable attributes, i.e., attributes whose values change (frequently) over time because of the normal operation of the system. For instance, the location attribute, which describes the physical location of the subject, changes every time the subject moves from one place to another.

Hence, for the enforcement of Usage Control policies, it is crucial that Usage Control systems are able to promptly retrieve the updated values of the set of attributes required to evaluate such policies to determine the related access decisions. In fact, the value of mutable attributes must be controlled both at request time and continuously during the execution of an access, in order to check whether the policy is still satisfied or it is violated every time these values changes. Summarizing, being able to promptly retrieve the updated value of the set of attributes exploited in the policy during the execution of an access is a crucial requirement for the effective enforcement of Usage Control policies.

However, an attribute provider, called Attribute Manager, could be (temporary) not available when the Usage Control systems request the values of the attribute it manages, and this could affect the usage decision processes. In fact, some access requests could be denied and some ongoing accesses could be revoked because the evaluation of the corresponding policies do not allow these accesses due to the missing attribute values.

To address this issue, this paper is focused on attribute retrieval in the Usage Control scenario, and it proposes a collaborative approach where a group of Usage Control systems cooperate to estimate the current attribute values when the corresponding Attribute Managers are not responding. The main idea is that a set of Usage Control systems are exploiting the same attributes to evaluate their Usage Control policies. When the corresponding Attribute Manager does not respond (e.g., the related server is unreachable or down), the Usage Control system which needs to retrieve the related attribute values contacts all the other Usage Control systems participating to the collaborative system to collect the values they cached for that attribute. We propose a strategy which allows to make a prediction of the current value of an attribute based on the values received from the other Usage Control systems. The contribution of this paper is summarized in the following:

- We propose a collaborative, distributed and decentralized model where different Usage Control systems cooperate to increase the availability of attributes provided by faulty Attribute Managers, exploiting cached copies of
the attributes stored by the other Usage Control systems that exploit the same attributes.

- We defined a quantitative reliability model for attribute values based on the probability of attribute change and on the reputation of the Usage Control Systems proving cached attribute values.
- We performed a set of simulative experiments to validate the proposed approach.

The remainder of the paper is organized as follows. Section II reports a motivating example showing a possible use case where the proposed model could be exploited to improve system performances. Section III recalls the background concept of the usage control model. Section IV describes the proposed framework, also introducing the theoretical aspects for attribute freshness, agents reputation and attribute validity (reliability) calculation, and Section V validates our framework showing a set of experimental results. Finally, Section VI reports some related work, while Section VII concludes the paper.

II. Motivating Example

As a motivating example, we consider a metropolitan environment where cars of distinct (at least two) car-sharing service providers have to comply to law specification concerning the level of pollution. Figure 1 gives a graphical representation of the logical architecture of the envisioned system. In particular, we suppose cars to have hybrid-engine technology, where the electric engine does not cause pollution but has limited performance (e.g., speed up to 40 km/h, fast discharge/slow charge), whilst the combustion engine has better performance (e.g., top speed much higher than city speed limit, longer autonomy, gas refill much faster than battery charging), still it produces emissions which increase the level of air pollution. Cars are allowed to use the combustion engine only if the current level of pollution is lower than a predefined threshold. The current level of pollution is measured by sensors, distributed inside the city, where each sensor is responsible for a specific area. Thus, when a car wants to turn-on the combustion engine, a usage control system installed in the car will query the sensor responsible for the area where the car is circulating, to receive the current pollution level. Hence, during the car ride, the Usage Control system determines when it is possible to use the combustion engine. However, pollution level sensors may temporary fail for several reasons. For instance, a solar battery sensor could fail because of a battery outage, but it will be available again as soon as the solar cell have recharged the battery. Other issues which may cause sensor unavailability could be high network congestion, or high electromagnetic interference. In this time interval, the current level of pollution cannot be retrieved. In such a case, when a car enters the area served by the faulty sensor, the safe-side strategy will suggest to deny the right of using the combustion engine, thus forcing this car to use only the electric one. Still, considering that the air pollution level is not generally supposed to drastically change in a matter of minutes, another car in the same area may have read a value of pollution before the sensor failure, which is still valid. This car shares the value of the air pollution with the other cars in the same area, using a 3G/4G Internet connection, which is also used to communicate with the attribute manager. Depending on the freshness of such a value, the first car decides either to consider this value as reliable or not. This operation brings however a risk. In fact, if a car turns-on the combustion engine when the pollution level is over the threshold, the company owning the car will receive a fine. On the other hand, denying the driver from using an higher performance engine when the pollution level is lower than the threshold, it is likely to cause user dissatisfaction, who may decide to migrate to another car sharing service provider. Another aspect to be considered is the possibility of unfair competition between different providers. In fact, knowing the two aforementioned risk factors, a provider may try to exploit them maliciously to cause a monetary loss to the competitor(s). To this end, this malicious service provider can decide to answer with false values for the pollution level, when the request comes from a competitor’s car.

III. Background: The Usage Control Model

The Usage Control (UCON) model [10], [12], [5] extends traditional access control models introducing mutable attributes and new decision factors besides authorizations: obligations and conditions. Mutable attributes represent features of subjects, resources, and environment that change their values as a consequence of the normal operation of the system [11]. For instance, some mutable attributes change their values because the policy includes attribute update statements that are executed before (pre-update), during (on-update), or after (post-update) the execution of the access. For instance, the e-wallet balance is a subject attribute which could be decreased by the policy every time the subject performs a new access to a resource. Other mutable attributes change their values because of actions performed by the subjects. For instance, the physical location of the subject is a mutable attribute of
the attributes required for the policy evaluation. For instance, system is exploited requires its own set of AMs to manage current values. Each specific scenario where the Usage Control retrieve and, in case of mutable Attributes, to update their Providers or Attribute Stores, which provide the interfaces to by Attribute Managers (AMs), sometimes called Attribute managers which are continuously evaluated while the access is in progress. Obligations are predicates which define requirements that must be fulfilled before the access (Pre-Obligations), or that must be continuously fulfilled while the access is in progress (ongoing-Obligations). Conditions are requirements that evaluate the attributes of the environment (e.g., current time or current workload). In this case too, pre-Conditions are evaluated when the subject requests to access the object, while ongoing-Conditions are continuously evaluated while the access is in progress. Obligations are predicates which must be fulfilled before the access (Pre-Obligations), or that must be continuously fulfilled while the access is in progress (ongoing-Obligations). Conditions are requirements that evaluate the attributes of the environment (e.g., current time or current workload). In this case too, pre-Conditions are evaluated when the subject requests to access the object, while ongoing-Conditions are continuously evaluated while the access is in progress.

The continuous evaluation of the policy when the access is in progress is aimed at interrupting the access when the execution right is no more valid, in order to reduce the risk of misuse of resources. Hence, in the Usage Control model it is crucial to be able to continuously retrieve the updated values of the mutable attributes, in order to perform the continuous evaluation of the policy and to promptly react to the attribute change by interrupting those ongoing accesses which are no longer authorized.

A. Attribute Retrieving in Usage Control Systems

For the sake of simplicity, this paper takes into account Usage Control systems based on the XACML reference architecture [9], with particular reference to the one presented in [4], which is shown in Figure 2. However, we think that adopting a different architecture should not prevent the adoption of the proposed approach.

In the XACML reference architecture, the Policy Enforcement Points (PEPs) embedded in the controlled system intercept the execution of a security relevant operation, and they invoke the Context Handler (CH), which is the frontend of the Usage Control system. The Policy Information Points (PIPs) are the components invoked by the CH to retrieve the attributes required by the Policy Decision Point (PDP) for the execution of the decision process, i.e., to evaluate the policy retrieved from the Policy Store (PS). Attributes are managed by Attribute Managers (AMs), sometimes called Attribute Providers or Attribute Stores, which provide the interfaces to retrieve and, in case of mutable Attributes, to update their current values. Each specific scenario where the Usage Control system is exploited requires its own set of AMs to manage the attributes required for the policy evaluation. For instance, LDAP servers or SQL databases could be exploited in some scenarios as AMs.

Hence, PIPs are properly configured in order to be able to query the specific AMs adopted in the scenario of interest for retrieving and updating attributes. In particular, each PIP implements the specific protocol required to interact with the related AM and exploits the provided mechanisms for securing the communications. The Usage Control model emphasizes the role of PIPs because it introduces the continuous policy enforcement while an access is in progress to cope with mutable attributes. In particular, the PIP is also in charge to detect when the value of an attribute changes in order to trigger the policy re-evaluation for the involved ongoing accesses, which are managed by the Session Manager (SM). To detect attribute changes, the PIP could exploit the subscription mechanism provided by the AM or the PIP must emulate it if it is not supported by the AM. For instance, the PIP could periodically retrieve the current attribute value in order to detect whether it is different from the previously collected one.

Moreover, other features can be implemented by PIPs to deal with issues with the AMs in some specific scenarios, such as the one proposed in this paper.

IV. FRAMEWORK MODEL AND WORKFLOW

This section reports a detailed description of the proposed framework and of its workflow.

Our reference scenario consists of a set of collaborating Usage Control Systems (UCSs). Each UCS follows the architecture reported in Section III, having thus one or more PIP for communication with one or more AMs. For the sake of simplicity, and without losing in generality, we assume that the faulty AM is only one, queried by all the UCSs in the reference scenario. The UCSs query the AM through a dedicated PIP to retrieve the value of an attribute which is required by the
UCS to evaluate the policy. However, the AM is considered faulty, i.e. it is possible that the PIP cannot communicate with the AM and the attribute cannot be collected. In this case, the UCS performing the attribute request forwards the request to the other UCSs in the system, asking for a cached version of the attribute. Each UCS can choose to ignore the request, or to effectively answer with a value for the attribute and the time at which the attribute has been read. Each UCS is completely autonomous and independent from the other UCSs in the system, i.e. the only thing that relates them is that they use the same attribute from the same AM. To this end a set of UCSs decide to cooperate to increase the availability of the commonly needed attribute. Being this cooperation completely P2P, it is possible that one of the UCSs decides to provide a false value for the attribute, as argued in Section II. We introduce in our framework a reputation based mechanism to control the reputation of each UCS on the base of the information it provides. To this end a Reputation Manager is added to the system. This component is responsible of storing the reputation level of all UCSs, updating them according to a Jossang-inspired reputation model which will be detailed in the following.

Table I schematically depicts the four main workflow steps of the collaborative attribute retrieval system. In the first step a UCS \((UCS_i)\) queries the AM for an attribute, finding it to be offline. Thus (step 2) \(UCS_i\) sends the attribute request to four other UCSs belonging to the cooperative system. The P2P communication between the UCSs may be enabled by an application installed on every UCS, which exposes a dedicated interface waiting for attribute requests. In fact, every UCS that wants to participate to the cooperative attribute retrieval needs to install this application. In step 3, the UCSs will respond with the couple of values \((x_i,f_i)\), representing respectively the value for the queried attribute and the time to which the it has been read from the AM. The different colors of the UCSs in Table I represent their behavior at step 3. The light blue cars \((UCS_1,UCS_3)\) provide a correct and fresh value for the attribute, the orange UCS \((UCS_2)\) provides a correct still not fresh value of the attribute, finally the red UCS \((UCS_4)\) provides a malicious value for the attribute. An attribute value provided by a UCS is considered correct if it is actually the last value retrieved from the AM by that UCS. In any other case, the provided value is considered malicious. A provided value is considered fresh if it is very unlikely that the real value is different. This probability depends on the specific attribute and on the frequency it normally varies its value. Additional details on the computation of such a probability is discussed in the following. As shown, in Step 3 \(UCS_R\) also retrieves from the reputation manager, the current value for the reputation of the four UCSs that sent the last read value for the required attribute. In step 4, \(UCS_R\) selects the attribute value, amongst the received ones, that will use for the policy evaluation, according to the following algorithm:

1) Three thresholds are set, for reputation \(\theta_r\), freshness \(\theta_f\) and validity \(\theta_v\), respectively;
2) The attributes values coming from UCSs having a reputation lower than \(\theta_r\) are discarded.
3) The attributes value whose freshness is lower than \(\theta_f\) are discarded.
4) From the remaining readings, the attribute value is chosen through a weighted sum (Eq 1), considering freshness and reputation for the provided values.
5) For the chosen value, \(UCS_R\) computes the reliability of the attribute value to evaluate the policy according to Equation 2, discarding the value if the result is lesser than \(\theta_r\).
6) Finally the reputation manager, according to all the received readings, i.e. considering also the one rejected by the system, updates the UCSs reputation, based on the provided reading. This computation is performed at system level independently from the identity and reputation of \(UCS_R\).

In the following subsections a formal description of the quantities and algorithms used to perform the aforementioned decision process is presented.

A. System Description

The collaborative system is composed of a set Usage Control Systems \(\mathcal{U} = \{U_1, U_2, \ldots, U_n\}\) which try to retrieve one or more attributes \(\mathcal{A} = \{A_1, A_2, \ldots A_m\}\). Each attribute \(A_i\) is provided by an AM which has a probability of being not available when a UCS attempts to retrieve the attribute. A feature paired with each attribute is its probability to change over time, i.e. given a specific time interval \(T\), it is known the probability \(P_{r,A_i}\) that the attribute \(A_i\) changes its value one or more times during the time interval \(T\).

If the attribute cannot be retrieved when an UCS, say \(UCS_R\), queries the AM for the value on \(A_i\), all other UCSs \(\mathcal{U}/\{UCS_R\}\) are asked if they own a cached copy of the value \(x\) of the \(A_i\) attribute. Each UCS \(U\) can either:

- Provide the last cached value of \(A_i\): \(x_{i,j}\);
- Do not provide a cached value for \(A_i\);
- Provide a malicious value for \(A_i\).

As discussed, it is considered malicious a value for the attribute \(A_i\) which is different from the last cached copy of the providing usage control system.

As an additional parameter, we assign to each usage control system a reputation score, based on the UCS behavior, considering the amount of non-malicious reads provided to the system.

B. Attribute Freshness

The freshness is an estimation of the probability that a specific attribute has not changed its value, since the last time it was read. Several models can be used to estimate such a probability, which mainly depends from the attribute type, the possible values it may assume and the update frequency. In the present work we will use a Poisson distribution to model this probability, which is effective in modeling a sequences of events that happen randomly and independently with a fixed rate over time [1]. Let consider \(\tau\) as the time that the next
event occurs in a Poisson process with the change rate $\lambda$. Thus, the probability density function for $\tau$ is defined as following [1]:

$$f_{\tau}(t) = \begin{cases} \lambda e^{-\lambda t} & t > 0 \\ 0 & t \leq 0 \end{cases}$$

In our context we consider a Poisson probability function for each attribute $A_i$, depending to its change rate $\lambda_i$. For instance, an attribute may change once a day ($\lambda = 1$), whilst the other may change once every 30 minutes ($\lambda = 48$).

For our analysis we consider that all the attributes are synchronized in the interval of time $[0, T]$. For instance, the interval of time can be considered 24 hours (1440 minutes), i.e. $T = 1440$. With the use of above probability density function, the probability that the attribute $A_i$ with the change rate $\lambda_i$ will change in the interval $(0, t]$ is equal to:

$$p(\tau \leq t) = \int_0^t f_{\tau}(t) \, dt = 1 - e^{-\lambda_i t}$$

Since the attribute $A_i$ is not synchronized in the interval $(0, T)$, the attribute $A_i$ may get out of date with the probability $p(\tau \leq t) = 1 - e^{-\lambda_i t}$ at time $t \in (0, T)$. Hence, the expected freshness is equal to:

$$E[F(A_i, t)] = 0 \cdot (1 - e^{-\lambda_i t}) + 1 \cdot e^{-\lambda_i t} = e^{-\lambda_i t} \quad t \in (0, T)$$

It is noticeable that the expected freshness is 1 at the time $t = 0$ and it approaches to 0 as the time passes. It closes to 0 more rapidly for the attributes which are updated more frequently. UCS considers as non fresh any value for which the freshness is lesser than the threshold $\theta_f$, which is a configurable parameter for each UCS.

To compute the expected freshness, it is required to have $t \in [0, T]$. To this end, let the current time be $t^*$ and the time that the user $j$ reads the attribute $A$ is denoted by $t_j(A)$. Then, the expected freshness of attribute $A$ provided by user $j$ at time $t^*$ is computed as follows:

$$f_j(A, t^*) = E[F(A, (t^* - t_j(A)) \mod T)]$$

where in the case that user $j$ read the attribute $A$ at almost the same time of request ($t^*$), then $(t^* - t_j(A)) \mod T \to 0$ and consequently $f_j(A, t^*) \to 1$.

For example, if the user send his request at time $t^* = 17:30 = 1050(m)$ and the $j$'th user has received the information at time $t_j(A) = 17:00 = 1020(m)$, then $f_j(A, t^*)$ is calculated as follows:

$$f_j(A, 1050) = E[F(A, (1050 - 1020) \mod 1440)] = E[F(A, 30 \mod 1440)] = E[F(A, 0)] = 1$$

Thus, the expected freshness is 1, which means that the attribute is fresh.
\[ t_j(A) = 17 : 20 = 1040(m) , \text{ then } t_j = 10. \]

C. Reputation

Since UCSs might be interested in providing a false (malicious) value for the attribute (as discussed in Section II), the introduction of a control mechanism is necessary. Considering that the envisioned collaborative system is completely Peer-To-Peer (P2P), the lack of a root of trust makes necessary the usage of a reputation-based model. The reputation model exploited in this work is based on the Jøsang model described in [2]. This model is based on a reputation score which is weighted by three components, namely belief, disbelief and uncertainty. The rationale behind choosing this specific model is the correspondence with the three possible actions that an UCS may perform when asked to provide a cached value. Every UCS has at the beginning a starting reputation score \( r_0 \). At each attribute reading in which a specific usage control system is involved, its reputation is updated according to the following formula:

\[
r(t) = b(t) - d(t) - u(t)
\]

where \( t \) is the time that the reputation is requested. Afterward, on the base of the provided value, after the decision process performed at step four of the system evolution, the three reputation component are updated according to the following algorithm.

**Algorithm 1 Updating reputation \( r_i \)**

\[
\begin{align*}
    r_i &= b_i - d_i - u_i \\
    b_i + d_i + u_i &= 1 \\
    \text{for all } u \in U_j \text{ do} \\
    &\text{if } u \text{ provides a correct value then} \\
    &\quad b_i = b_i + \Delta_b \\
    &\quad u_i = u_i - \Delta_u \\
    &\quad d_i = d_i - \frac{\Delta_u}{3} \\
    &\text{else} \ \\
    &\text{if } u \text{ provides a non fresh value then} \\
    &\quad u_i = u_i + \Delta_u \\
    &\quad b_i = b_i - \Delta_u \\
    &\text{end if} \ \\
    &\text{else} \ \\
    &\text{if } u \text{ provides a malicious value then} \\
    &\quad d_i = d_i + \Delta_d \\
    &\quad b_i = b_i - \Delta_d \\
    &\quad u_i = u_i - \frac{\Delta_d}{3} \\
    &\text{end if} \\
    &\text{end if} \\
\end{align*}
\]

The belief component is increased every time the UCS provides a value that is not malicious, i.e. the provided value is the one effectively considered good by UCS\(_R\). If the provided value is considered not fresh, the uncertainty component is increased, whilst the disbelief component is increased if the provided value is considered fresh but malicious. No reputation changes happen for those UCS that choose to not provide any value. The values of \( \Delta_d, \Delta_u \) and \( \Delta_b \) are configurable parameters. For the experiment performed in this work, the used values are: \( \Delta_b = 0.25, \Delta_u = 0.15, \Delta_d = 0.6 \), whilst the acceptance threshold for reputation \( \Theta_r \) is set to 0.5. These values mildly increase at each reading the reputation of those users providing correct values, strongly penalize the UCSs providing a malicious value, immediately reducing their reputation under the acceptance threshold. The uncertainty has a small impact on the reputation, which becomes consistent only after several non-fresh readings.

In order to incentive a fair cooperation, the UCSs participating to the system will receive a reward, which is proportional to the current reputation. This reward can either be based on money or additional services, depending from the specific application. We refer the interested reader to some examples of reward models which are discussed in [7].

D. Attribute Validity

With the definitions of freshness and reputation, it is now possible to formally define the decision process, used by UCS\(_R\) to choose the value \( v^* \) amongst the ones \( \{v_1, v_2, \ldots, v_s\} \) provided for attribute \( A \). For instance, the attribute for computing the air pollution may contain its domain as \( A = \{positive, negative\} \), such that positive means that the level of air pollution is higher than (or equal to) the specified threshold and negative vice versa.

Let’s consider that the request from UCS\(_R\) happens at time \( t \). Moreover, let \( \Theta_f \) and \( \Theta_r \) be the thresholds of attribute freshness and reputation respectively. To make the final decision, i.e., to choose the best value for \( A \), the system collects the triples \( U_j(A,t) = \{v_j(A,t), f_j(A,t), r_j(t)\} \), for each user who participated, where \( U_j \) is the \( j \)’th user, \( v_j(A,t) \) is the value of attribute \( A \) provided by \( j \)’th user at time \( t \), \( f_j(A,t) \) returns the expected freshness of \( v_j(A,t) \) provided by user \( j \), and \( r_j(t) \) is the reputation of \( U_j \) at time \( t \). The system discards the triples for them either \( f_j(A,t) \leq \Theta_f \) or \( r_j(t) \leq \Theta_r \). After discarding not fresh and not reliable values, the set of \( n \) triples considered is defined as follows:

\[
U_j(A,t) = \{v_j(A,t), f_j(A,t), r_j(t)\} \quad \text{for } 1 \leq j \leq n \quad (1)
\]

Then from the collected information at time \( t \) (relation (1)), we calculate the validity of the \( k \)’th value of attribute \( A \), i.e. \( v_k \), denoted by \( V(v_k,t) \), as follows:

\[
V(v_k,t) = \frac{1}{n} \sum_{j=1}^{n} \delta_j(v_k,t) \times f_j(A,t) \times r_j(t) \quad (2)
\]

Moreover, \( \delta_j(v_k,t) \) is computed as follows:

\[
\delta_j(v_k,t) = \begin{cases} 
  1 & \text{ if } v_k = \pi_1(U_j(A,t)) \\
  0 & \text{ Otherwise}
\end{cases}
\]

where \( \pi_1(U_j(A,t)) \) refers to the first component of \( U_j(A,t) \). The result of \( \delta_j(v_k,t) \times f_j(A,t) \times r_j(t) \) equals to 0, or approaches to 0, if:

1) the \( j \)’th user reports a value of \( A \) except \( v_k \) at time \( t \), i.e. \( \delta_j(v_k,t) = 0 \).
2) the freshness of attribute $A$ provided by $j$’th user at time $t$ approaches to 0, i.e. $f_j(A,t) \to 0$.
3) the reputation of $j$’th user equals to 0, or approaches to 0, i.e. $r_j(t) = 0$ or $r_j(t) \to 0$.

The system returns the value $v^*$ of $A$ that returns the maximum amount of validity, i.e.

$$v^* = \arg \max_{v_1 \leq v_2 \leq v_j} V'(v_k,t)$$

(3)

V. Results

In order to evaluate the effectiveness of the proposed collaborative system, a set of simulative experiments has been performed. The simulation framework simulates a series of subsequent reading cycles, in which the AM is not available, one UCS, $UCS_R$, needs to retrieve the value of an attribute managed by the AM, and all other UCSs are queried for a cached value of the required attribute. For the sake of simplicity, we only consider two possible values for the attributes, i.e. correct or malicious. We have set up a testbed consisting of 100 agents participating to the collaborative system. At each reading cycle, we randomly choose one of the agents to be the $UCS_R$, whilst all the others may provide or not the values they already read for that attribute, as discussed in the previous sections. The number of agents that choose to omit to provide their values is a parameter of the simulation and, at each reading cycle, the UCSs omitting to provide attribute values are chosen casually. Other parameter of the simulation framework is the percentage of malicious agents. The malicious UCSs provide a malicious value of the attribute with a probability of 50%. The rationale of this behavior is to model both the possibility that a malicious UCS tries to keep a low profile, alternating malicious behaviors to genuine ones, and to consider the possibility that a malicious agent sends genuine reports toward selected UCSs. As parameters for this simulation, we have set the reputation and the freshness threshold, respectively, to $\theta_r = 0.5$ and $\theta_f = 0.5$. For the sake of simplicity, in this set of experiments we assume that malicious users do not alter the freshness of their readings.

Figure 3 shows the evolution of reputation for genuine and malicious agents for a series of 100 reading cycles, with the aforementioned values for the reputation algorithm, and varying the percentage of malicious UCSs in the system in the interval $[10, 50]$. We assume that at each reading 20 UCSs out of 99 choose to omit to provide their values. As can be seen, regardless of the percentage of malicious agents, when the number of reading cycles increases, the reputation of good agents rapidly increases to 1, whilst the reputation of malicious agents never goes above the reputation threshold $\theta_r$. This result demonstrates how the adopted reputation model is effective in our scenario in preventing malicious UCSs from having any negative effect on the system. In our experiments, we do not consider a percentage of malicious agents greater than 50% because, in this case, in any distributed system, if the malicious agents are colluding, it is not possible to control the system neither through voting nor reputation [2]. For an additional insight, Table II reports additional details concerning the experiment in which the percentage of malicious UCSs is fixed at 50%.

<table>
<thead>
<tr>
<th>UCS Behavior</th>
<th>Total</th>
<th>Correct</th>
<th>Malicious</th>
<th>Fresh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Good</td>
<td>80</td>
<td>80</td>
<td>0</td>
<td>49</td>
</tr>
<tr>
<td>Bad</td>
<td>80</td>
<td>40</td>
<td>40</td>
<td>50</td>
</tr>
</tbody>
</table>

The table reports for the set of 100 reading cycles the details of the values provided in average by a Good and by a Malicious UCS in the collaborative system. In particular, the first column reports the average amount of values provided by UCSs, which is 80, in line with the simulation parameters, that at each reading cycle selects 80 of the 100 available UCSs. The second column reports the number of Correct values reported, amounting to 80 for good users and to 40 for bad agents. As a matter of fact, bad agent send a malicious value with a probability of 50%. The remaining 40 values, in fact, fill the third column representing the malicious value provided to the system. The fourth column reports the number of fresh values, i.e. those values for which the freshness is higher than the threshold $\theta_f$.

Figure 4 reports the maximum validity value (computed as described in Section IV-D) for each of the 100 reading cycles, with five different percentages of malicious UCSs. The red dotted line represents the validity threshold $\theta_v$. We recall that the value resulting from a reading cycle is effectively used by the $UCS_R$ only if the related validity is over the threshold line. As shown, with the used configuration the first reading always falls under the threshold, due to the selected starting reputation value. However, even if the value is discarded, the reputation of providing UCSs is updated and, starting from the second reading, our system is always able to find a value for the attribute $A$ whose validity is above the threshold, i.e., a value which can be effectively used by $UCS_R$.

VI. Related Work

Fault tolerance in distributed systems is a well known topic analyzed in several work like [8] and [13] which propose...
be collected at policy evaluation time. In this paper we have proposed a methodology to increase the attributes availability in systems where AM may be faulty for limited period of time. The proposed methodology is based on the collaboration of different UCSs accessing the same AMs, caching the value of the last reading, to share it with the other UCSs that are temporarily unable to contact the AM. Issues related to attribute freshness and reputation are also considered and evaluated with simulative experiments. As future works, we plan a deeper analysis of more complex systems including colluding attackers, as well as application in real settings for a more effective evaluation of the proposed system.

Acknowledgements

This work was supported by the EU FP7 project Confiden
tial and Compliant Clouds (CoCoCloud), GA #610853 and by the H2020 EU-funded project European Network for Cyber Security (NeCS), GA #675320.

References

[7] Flix Gmez Mrmol and Gregorio Martinez Prez. Towards pre-
standardization of trust and reputation models for distributed and heter-
[13] K. Ranganathan, A. Iamnitchi, and I. Foster. Improving data avail-

VII. Conclusion and Future Work

Many Access and Usage Control systems commonly adopted to enforce security policies on IT systems and on cyber-physical ones, are based on attributes whose values must...