A Formal Approach for the Verification of AWS IAM Access Control Policies

Ehtesham Zahoor, Zubaria Asma, Olivier Perrin

A Formal Approach for the Verification of AWS IAM Access Control Policies

Ehtesham Zahoor[1], Zubaria Asma[1], and Olivier Perrin[2]

[1] Secure Networks and Distributed Systems Lab (SENDS),
National University of Computer and Emerging Sciences, Islamabad, Pakistan
ehtesham.zahoor@nu.edu.pk, zubaria.asma@gmail.com
[2] Université de Lorraine, LORIA BP 239 54506
Vandoeuvre-lès-Nancy Cedex, France
olivier.perrin@loria.fr

Abstract. Cloud computing offers elastic, scalable and on-demand network access to a shared pool of computing resources, such as storage, computation and others. Resources can be rapidly and elastically provisioned and the users pay for what they use. One of the major challenges in Cloud computing adoption is security and in this paper we address one important security aspect, the Cloud authorization. We have provided a formal Attribute Based Access Control (ABAC) model, that is based on Event-Calculus and is able to model and verify Amazon Web Services (AWS) Identity and Access Management (IAM) policies. The proposed approach is expressive and extensible. We have provided generic Event-Calculus modes and provided tool support to automatically convert JSON based IAM policies in Event-Calculus. We have also presented performance evaluation results on actual IAM policies to justify the scalability and practicality of the approach.

Keywords: AWS Cloud, IAM, Access Control, Verification, Event-Calculus

1 Introduction

Information security has been in the mainstream of computing. In the last decade, advancements in the domain of Cloud computing have further amplified the need to protect digital information. Cloud computing offers elastic, scalable and on-demand network access to a shared pool of computing resources such as storage, computation and communication. Resources can be rapidly and elastically provisioned and the users pay for what they use. These benefits and offerings from different Cloud providers have improved its adoption as businesses are seeking new opportunities to reduce hardware and management costs by offloading their capabilities to the Cloud. One of the major challenges in Cloud computing adoption is security for Cloud users.

The security policy of an organization helps to better prepare for and address security challenges. It is a high-level specification of how to implement security principles and technologies. For instance, the Authentication policy of an organization specifies which users are allowed to use its services. In this paper we

address the issues related to one important class of security policies, called the Access Control or Authorization policies. There is an important distinction between the authentication and authorization policies of an organization. When a user attempts to access some resource, the first step is to determine and validate the identity of the user using some authentication measures such as login credentials. These credentials are then matched with the organization's authentication policy to identify the validity of user. Once a user has been authenticated, the authorization process involves determining what rights a user has. The authorization process allows to determine who can access what resources, under what conditions, and for what purpose. The authorization process can be based on temporal aspects and may involve delegation. While the Cloud based authentication has been a highly active research direction, Cloud authorization has remained relatively less explored. In this paper, we have provided a formal attribute based access control model, that is based on Event-Calculus and is able to model and verify authorization policies. Specifically our contributions include:

A formal authorization model: *In contrast to traditional XML (or JSON in case of AWS IAM) based authorization policy specification languages, our approach is formal and based on Event-Calculus, a logical language for specification of and reasoning about events and their effects.*
AWS IAM policies verification: *We have applied our approach to model and verify AWS IAM policies. We have categorized conflicts as either Intra or Inter-Policy conflicts. To best of our knowledge there exists no approach that attempts to model and verify IAM policies.*
ABAC based approach: *Our approach is based on Attribute Based Access Control (ABAC) and it is by design a generic approach to handle other authorization models. For instance, AWS IAM is based on Role Based Access Control (RBAC) and our proposed approach allows it to be modeled and verified by considering Role as an attribute.*
Extensible approach: *The proposed approach can be extended to model other Authorization services provided by Cloud providers. For instance, OpenStack provides Role-Based Access Control for networks (Neutron) and user management. Our approach can be used to formally verify and reason about them.*
Tool support and performance evaluation: *We have provided generic Event-Calculus models and provided tool support to automatically convert JSON based IAM policies in Event-Calculus. We have also presented performance evaluation results on actual IAM policies to justify the scalability and practicality of the approach.*

2 Background and related work

The term Access Control in the context of Cloud computing research has attracted interest in two broad subdomains. A number of approaches have addressed the security issues related to the data storage on the Cloud based storage services. In this context Attribute-Based Encryption (ABE) [1] has been

proposed which implements attribute-based access control by encrypting data based on attributes. In such a scheme, only authorized users having same set of attributes can decrypt the data. A number of approaches have been proposed to address different related aspects such as introduction of attribute hierarchies [2], handling of the attributes revocation problem [3], P2P storage Cloud [4] and attribute-based keyword search scheme with user revocation [5].

The other subdomain for the research related to access control includes the policy languages for specifying authorization policies. For Cloud based applications or resources, authorization should not only be performed based on the content, but also by the context and is prone to performance, bandwidth, attributes availability and other requirements. The authorization process and policies can be considered from the enterprise or federation point of view, using approaches such as XACML, or from a user point of view (e.g. OAuth or Lockr). In general, access control and authorization has remained an active research area and a basic approach is to assign access policy directly to end users. This approach however does not scale with the increase in number of users. A number of approaches thus consider the Role Based Access Control (RBAC) model and its variations [6]. In RBAC users are assigned roles and the access policy is associated with these roles. Task based access control (TBAC) extends the traditional model by considering task based contextual information. Even though RBAC is a well defined model and still being used extensively, for instance AWS IAM is RBAC, it suffers from *role explosion* as too many roles (may even surpass the number of users) may need to be managed [7]. Some approaches have investigated the use and challenges for RBAC in a distributed environment [8–11].

In contrast to RBAC models, the Attribute based Access Control (ABAC) model is based on the attributes [12]. The resources, subjects and environment have attributes and the policy rule is a boolean function on these attributes. ABAC model can be considered more generic and provides more flexibility and expressiveness than RBAC models. ABAC can subsume RBAC as a role itself can be an attribute in an ABAC model. XACML (eXtensible Access Control Markup Language) is a XML-based language based on the ABAC model. XACML is verbose and based on XML and this makes it difficult to analyze and verify the consistency of a set of policies. A number of approaches to provide formal semantics of XACML have been proposed [13–15]. Further, a number of approaches have been proposed that build upon XACML for its usage in collaborative and distributed environments. These include [16] in which the authors propose a distributed device access control architecture called *MPABAC*. In [17] the authors have developed a formal policy language *BelLog* that can express both delegation and composition operators. Some access control policies are user-centric, that is when the user determines the access for their resources. The most prominent approach being the OAuth [18] which allows users to share their personal resources with other sites without giving them their credentials. User-Managed Access (UMA)[3] is another user centric approach and it provides

[3] http://docs.kantarainitiative.org/uma/draft-uma-core.html

services for authorization, monitoring and changing data sharing. Lockr [19] is an access control system based on social relationships.

Formal methods are being used at Amazon to verify and validate their distributed systems since last few years [20]. They have used *TLA+*, a formal specification language based on basic set theory and predicates, and *PlusCal*, closer to a C-style programming language and even more expressive than *TLA+* to model and verify AWS services such as *S3*, *DynamoDB* and *EBS*. However, the AWS IAM Policies are not formally verified and although AWS does provide a *PolicySimulator*, its scope and usage is limited as it does not attempt to verify the consistency of policies . In this work we have used the Event-Calculus, a logic programming formalism, to model and verify AWS IAM policies. Our approach builds upon our previous work in handling temporal, trust and delegation aspects in distributed environments [21, 22]. In this work, we have thoroughly updated the models and instead of trust and/or temporal aspects considered AWS IAM policies verification. We have provided generic models, tool support and the performance is evaluated on actual AWS IAM polciies. To best of our knowledge there exists no approach, other than limited AWS Policy Simulator, that attempts to model and verify IAM policies.

3 AWS IAM Policies Specification

The Identity and Access Management (IAM) service provided by Amazon Web Services (AWS) is an example of RBAC model. The service provides both authentication and authorization. IAM has a notion of policy which is a high level representation of the actions a user is allowed to perform on resources, Figure 1. The policies are high level description that explicitly lists permissions. Each *policy* has a set of *statements* and on a broad level a *statement* specifies the following:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect":  "Allow",
            "Action":  ["ec2:*"],
            "Resource":["*"]
        },
        {
            "Effect": "Deny",
            "Action": ["ec2:*"],
            "Resource": ["*"]
        }
    ]
}
```

Fig. 1. An example AWS IAM policy with two statements

- **Service and Resources:** You can specify to which AWS service this policy applies; such as *Amazon EC2* or *Amazon S3*. Then for each service you can further specify to what specific resource this statement refers. Resources are specified using Amazon Resource Name (ARN).
- **Actions:** You can further specify to what specific action(s) this statement applies. The set of actions are service-dependent and each AWS service has its own actions, for instance the *CreateKeyPair* action is associated with *Amazon EC2* service. You can select all actions using the Policy Generator or use a wildcard (*) in the JSON document.
- **Effect:** You need to specify if the effect of the statement is either *Allow* or *Deny*. For instance you can specify that a statement allows some action on some resource of an AWS service.
- Conditions: You can optionally further constrain a statement by providing conditions which are specified by providing a *condition* (for instance *StringEquals*), a *key* (for instance *aws:userid*) and a *value*.

Each policy document is stored in JSON format, see Figure 1, and contains a set of statements, each at least having the elements mentioned above. A policy may contain other elements such as *statement ID (sid)* and policy version. Once a policy has been created, it can be assigned directly to IAM Users. This basic form of access control model can be termed as User Based Access Control as discussed in Section-II. This approach would not scale and would be hard to manage with the increase in number of users. Alternatively, AWS allows to assign a policy to IAM Groups, a collection of users. For example, you can create an IAM Group named, *Administrators*, assign it a policy giving complete access. You can then add and remove users from this group as the need arises. Such an access control model is termed as Role Based Access Control (RBAC). However, one major limitation associated with RBAC based models is Role Explosion. It may be feasible when the number of roles (IAM groups) is small but for large organizations the number of roles may eventually surpass the number of users. This is because of various reasons such as the scale of services provided by AWS, most having numerous resources such as number of Buckets in *S3*. In addition, a large number of actions can be performed on these services and their resources. Principle of least privilege would force policy designer to create numerous roles and it would make it difficult for this model to scale.

Then there are other limitations regarding policy specification and its verification as provided by AWS IAM. The conflicts in policy specification can be broadly categorized into intra-policy and inter-policy conflicts. Intra-policy conflicts are within a single policy while the inter-policy conflicts are when multiple policies are combined and attached to a single user or group. If we closely look at the policy specification in Figure 1, we can see that the two statements are conflicting; one allows for the access to EC2 while the other denies it. During policy specification, AWS does provide an option to validate the policy but it only checks if the policy is syntacticly correct and does not provide such conflict detection.

4 Proposed approach

The proposed approach for AWS IAM Policies modeling is based on Attribute Based Access Control (ABAC) model and this choice is both to address the scalability and role explosion limitations associated with the IAM RBAC model, as discussed in previous section. In contrast to RBAC model, the ABAC model is based on the attributes [12]. The resources, subjects and environment have attributes and the policy rule is a boolean function on these attributes. ABAC model can be considered more generic and provides more flexibility and expressiveness than RBAC models. ABAC can subsume RBAC as a role itself can be an attribute in an ABAC model. The proposed models build on our previous work on providing a formal approach to XACML [21]. The proposed policies specification approach is based on Event-Calculus modeling formalism.

The choice of Event-Calculus is motivated by several reasons. Space limitations restrict us to provide an exhaustive comparison of all temporal languages, however based on our analysis we do believe that Event-Calculus has many interesting properties to model access control policies. First, Event-Calculus integrates an explicit time structure, in contrast to Situation Calculus, and is independent of any sequence of events (possibly concurrent). A second advantage of using Event-Calculus (over Linear Temporal Logic for instance) is that Event-Calculus supports the possibility to express quantitative time constraints (unlike LTL, except considering extensions and with limitations – see extensions of CTL/LTL). Then, considering policies that could include intervals (for instance, an access policy is set from 8pm to 7am), the ability of Event-Calculus to handle intervals (e.g. Allen's intervals) is definitely interesting. Third, as underlined in [23], techniques based on LTL are not fully suitable for continuous support, whereas in our context, as events occur, the Event-Calculus models are able to detect possible violations of the policies as soon as an event is detected. It allows us for a number of reasoning tasks that can be broadly categorized into deductive, abductive, and inductive tasks. In relation to TLA+ we believe that the security policies are more event-driven and thus Event-Calculus is a better choice. Fourth, using Event-Calculus provides the ability to express constraints not only upon actions, but also on data. Last, Event-Calculus is very interesting as the same logical representation can be used for verification at both design time (static analysis) and runtime (dynamic analysis and monitoring).

4.1 Event-Calculus

Event-Calculus is a logic programming language ([24]), first proposed by Robert Kowalski and Marek Sergot in 1986. The event-calculus represents the effect of *Actions* on *Fluents*. Event-Calculus has a set of *events* (or actions) that trigger the change, $\mathcal{A}$, a set of *fluents* that represent anything whose value is subject to change over time, $\mathcal{F}$, a set of time points $\mathcal{T}$, and a set of objects related to the particular context $\mathcal{X}$. Some basic event calculus predicates used for modeling the proposed framework are:

- *Initiates*(e, f, t) - fluent f holds after timepoint t if event e happens at t.
- *Happens*(e, t) specifies that event e happens at timepoint t.
- *HoldsAt*(f, t) is true iff fluent f holds at timepoint t.

The Event-Calculus models are presented using the discrete Event-Calculus language [25] and we will only present the simplified models that represent the core aspects, intentionally leaving out the supporting axioms[4]. All the variables (such as *stmt*, *time*,. . .) are universally quantified. Due to space limitations, some names are either abbreviated. In addition, we have shortened representation of some events and fluents such as *AllowPolicy* and *DenyPoliy*, are written as *Allow/DenyPolicy*.

4.2 Statements specification

The *statements* (abbreviated as stmt in our models) allow to specify one specific access rule. Each statement has a *Target*, an *Effect* and the associated *Conditions*. This would seem different from the IAM policy model where statements contain other elements such as *Actions* and *Resources*. This approach is at the heart of our ABAC model as we treat all the information needed as to be composed of name-value attributes. For instance, the *Resource*, the *Action*, the *Group* of the user and other such information is considered as attributes having names and values. It thus allows for adding new attributes for target specification if needed. We start our Event-Calculus modeling approach by first presenting the Event-Calculus model for specifying statements and then using *DECReasoner*[5] to reason about a statement.

Statements Model 1 (Meta-model for IAM Statements)

```
;Sorts for attributes name/values
sort stmt, atname, atvalue      predicate AtHasValue (atname, atvalue)
;Fluents for Stmts evaluation
fluent StmtTargetHolds(stmt), StmtConditionHolds(stmt)
fluent StmtEffectIsPermit(stmt), StmtIsPermitted/Denied/NotApplicable(stmt)
;Events for Stmts evaluation
event (Mis)Match(stmt), Approve/DenyStmt(stmt), StmtDsntApply(stmt)

;These axioms link fluents with events
Initiates (Match(stmt), StmtTargetHolds(stmt), time).
Initiates(Approve/DenyStmt(stmt), StmtIsPermitted/Denied(stmt), time).
Initiates(StmtDsntApply(stmt), StmtIsNotApplicable(stmt), time).

;Conditions on events occurrence
Happens(ApproveStmt(stmt), time) -> HoldsAt(StmtTargetHolds(stmt), time) &
HoldsAt(StmtCondHolds(stmt), time) & HoldsAt(StmtEffectIsPermit(stmt), time).
Happens(StmtDsntApply(stmt), time) -> !HoldsAt(StmtTargetHolds(stmt), time).

;Initial state of the Fluents
!HoldsAt(StmtIsPermitted/Denied/NotApplicable(stmt),0).
;The goal for the reasoner
HoldsAt(StmtTargetHolds(stmt),1) | !HoldsAt(StmtTargetHolds(stmt),1).
HoldsAt(StmtIsPermitted/Denied/NotApplicable(stmt),2).
```

[4] Complete models can be found at https://members.loria.fr/operrin/files/esocc.txt
[5] http://decreasoner.sourceforge.net/

In the Event-Calculus model above, we first define some sorts, such as *stmt*, *atname* and *atvalue*, which can be considered as types of which individual variables can be instantiated. We use the sort named *stmt* to represent individual statements. Similarly the sorts *atname* and *atvalue* would be used to model attribute names and value respectively. We have then defined a predicate *AtHasValue* which specifies name-value pairs for attributes.

The core part of the model above concerns definition of fluents and events to model the state of a statement being evaluated. A fluent is anything whose value is subject to change over time and we have thus defined fluents such as *StmtIsPermitted/Denied/NotApplicable*. A statement is neither *Approved*, *Denied* or *NotApplicable* by default so the fluents are initialized such that they do not hold at the start. We then define some events which can happen and whose occurrence would change the fluent state. To link an event with fluent state, we use Event-Calculus initiates axioms and for instance, if the event *ApproveStmt* happens at time *t*, the fluent *StmtIsPermitted* would hold at timepoint *t+1*. Then we have defined some constraints on events occurrence; for instance *ApproveStmt* event can only happen at time *t*, if the fluents *StmtTargetHolds*, *StmtCondHolds* and *StmtEffectIsPermit* holds. Finally we specify the initial conditions for the fluents and the goal for the reasoner. The *Match/Mismatch* events occurrence decide if the fluent *StmtTargetHolds* holds or not. If the *StmtTargetHolds* doesn't hold, we consider the statement to be not applicable, *StmtIsNotApplicable*. If the statement does apply, that is fluent *StmtTargetHolds* holds, it would decide if the statement is permitted or denied based on its conditions and effects.

The model above has been intentionally made generic and can be considered as a meta-model. We can put this model in a file and include the file for the specification of any specific statement. As an example on how to use the generic model, we model the IAM policy statement, as shown in Figure 1, which allows any action on any EC2 resource.

Statements Model 2 (AWS IAM statement specification)

```
load includes/stmts/... ;generic model
atname Object, Action
atvalue AnyEC2resource, AnyAction
AtHasValue(Object,AnyEC2resource). AtHasValue(Action,AnyAction).
stmt StmtAllow

;Specifying when the statement target holds
Happens(Match(stmt),time) & AtHasValue(Object, atvalue1) & AtHasValue(Action,
atvalue2) -> atvalue1 = AnyEC2resource & atvalue2 = AnyAction.

HoldsAt(StmtEffectIsPermit(StmtAllow),0).
HoldsAt(StmtConditionHolds(StmtAllow),0).
```

In the model above, we instantiate the generic model for a specific IAM statement. We first thus include the generic model files and then specify attribute names/values and link them using a predicate *AtHasValue*. We name the statement (by creating an instance of sort *stmt*) as *StmtAllow*. Then we define a conditional axiom that the event *Match* can only happen if the attribute name value pairs match (we define the same for *Mismtach* event but is not shown due

to space limitations). If we invoke the Event-Calculus reasoner, called *DECReasoner*, for the Event-Calculus based specification, it returns a solution as shown below.

Solution 1 (Statement evaluation using DECReasoner)

```
55 variables and 163 clauses
relsat solver
1 model
—
model 1:
0
StmtConditionHolds(StmtAllow).
StmtEffectIsPermit(StmtAllow).
Happens(Match(StmtAllow), 0).
1
+StmtTargetHolds(StmtAllow).
Happens(ApproveStmt(StmtAllow), 1).
2
+StmtIsPermitted(StmtAllow).
```

The solution returned by *DECReasoner* is shown above. In order to reason about Event-Calculus models, *DECReasoner* first encodes the problem in a Satisfiability (SAT) problem and then invokes the SAT solver, to reason about the models. The solution shows that the encoded SAT problem has 55 variables and 163 clauses. Then for each time-point, the solution shows which events happen at that time-point and what fluents hold true at that time-points. In case a fluent starts to hold true at time-point t (after an event happens at time-point *t-1*) it is shown with a plus(+) sign. The solution above shows that as the attributes' values are intentionally same as the ones specified in the statement, the statement target thus holds. If we change any of the attributes like the *Resource* has any other value, the *DECReasoner* will provide a model which shows that the event mismatch would happen and the statement does not apply to it, modeled by the fluent *StmtIsNotApplicable(stmt)*.

Once the target of the statement holds, it is then evaluated based on associated *Condition* and *Effect*. The statement Effect is to either *Permit* or *Deny* and the rule Condition can be considered as a set of predicates, based on the functional and the non functional constraints, that specify what conditions we need to check for the statement. In the statement above, we intentionally considered statement effect to be *Permit* modeled by fluent *StmtEffectIsPermit(StmtAllow)*, and the condition to hold, modeled by fluent *StmtConditionHolds(StmtAllow)*.

5 Intra-Policy Conflicts

For the proposed approach, individual statements can be grouped into a policy, similar to the IAM policy. The proposed modeling approach is generic and thus allows for easily aggregating statements. In order to discuss the Event-Calculus models related to policies, let us consider that another statement named *StmtDeny* exists which is similar to the *StmtAllow* but having effect as Deny (space limitations restrict us to detail the model). The proposed policy Event-Calculus model is shown in the model below:

Policy Model 1 (Meta-model for IAM Policies)

```
sort policy        predicate PolicyHasStmt(policy, stmt)
;Fluents for Policy State/Evaluation
fluent PolicyIsPermitted/Denied/NotApplicable/Invalid(policy)

;Events for Policy State Change
event PolicyDoesntApply(policy), Approve/Deny/InvalidatePolicy(policy)
;Initiates Axioms for Events/Fluents
Initiates(PolicyDoesntApply(policy), PolicyIsNotApplicable(policy), time).
Initiates(Approve/DenyPolicy(policy), PolicyIsPermitted/Denied(policy), time).
Initiates(InvalidatePolicy(policy), PolicyIsInvalid(policy), time).

;Policy is invalid if the outcome of stmts is conflicting
Happens(InvalidatePolicy(policy), time) -> {stmt1, stmt2}PolicyHasStmt(policy, stmt1)
& PolicyHasStmt(policy, stmt2)
& HoldsAt(StmtIsPermitted(stmt1), time) & HoldsAt(StmtIsDenied(stmt2), time) .
;Initial conditions for fluents
!HoldsAt(PolicyIsPermitted/Denied/NotApplicable/Invalid(policy)(policy),0).
```

5.1 Statements combining Algorithms

The proposed approach does not only allow for conflict detection but rather is generic to model other combination algorithms. For instance, the *Permit Overrides* would permit a Policy in case of conflicting outcome of statements and *Deny Overrides* (the only option currently provided by AWS IAM) would deny a policy in case of any statement being Denied. The choice of expressive Event-Calculus allows a number of other combining algorithms based on temporal, cardinality (for instance decision is based on majority x out of y statements), trust and other aspects. Space limitations restrict us to detail them further.

Policy Model 2 (Meta-model for IAM Policies - Combining Algorithms)

```
;Permit if even one of the stmts is permitted - permit overrides
Happens(ApprovePolicy(policy), time) -> {stmt} PolicyHasStmt(policy, stmt) &
HoldsAt(StmtIsPermitted(stmt), time).
;Deny if all of the stmts are denied
Happens(DenyPolicy(policy), time) & PolicyHasStmt(policy, stmt) ->
HoldsAt(StmtIsDenied(stmt), time).
```

5.2 Instantiated Policy Model

In order to see an example of intra-policy conflicts identification, we instantiate the generic Policy model shown above to model the policy shown in Figure 1.

Policy Model 3 (AWS IAM Policy Specification)

```
;Load generic models for statements/policies and instantiated statements
load includes/stmts/...      load includes/policy/...
load includes/stmts/defined/StmtAllow/StmtDeny.e

policy AmazonEC2FullAccess
PolicyHasStmt(AmazonEC2FullAccess, StmtAllow/StmtDeny).
;Goal: Decide if the policy is permitted/denied/NotApplicable/Invalid
HoldsAt(PolicyIsPermitted | Denied | Invalid...(policy),3).
```

In the model above, we have already defined two statements, *StmtAllow* and *StmtDeny* and we add them to a policy using the predicate *PolicyHasStmt*. The result returned by the *DECReasoner* is shown below. As both the statements concern the attributes event *Match* happens for both statements. As the effect of one statement is *Permit* and other is *Deny*, so at time-point 2, one gets permitted and other gets denied (as shown by fluents *StmtIsDenied* and *StmtIsPermitted*). Then at time-point 2, event *InvalidatePolicy* happens and the policy is considered invalid.

Solution 2 (Policy evaluation result by DECReasoner)

```
0
StmtConditionHolds(StmtAllow).
StmtConditionHolds(StmtDeny). StmtEffectIsPermit(StmtAllow).
Happens(Match(StmtAllow), 0). Happens(Match(StmtDeny), 0).
1
+StmtTargetHolds(StmtAllow). +StmtTargetHolds(StmtDeny).
Happens(ApproveStmt(StmtAllow), 1). Happens(DenyStmt(StmtDeny), 1).
2
+StmtIsDenied(StmtDeny). +StmtIsPermitted(StmtAllow).
Happens(InvalidatePolicy(AmazonEC2FullAccess), 2).
3
+PolicyIsInvalid(AmazonEC2FullAccess).
```

The proposed intra-policy conflicts verification approach provides a number of benefits. First the proposed models are intentionally made generic and thus it is easy to model policies and statements, without going into concrete details of Event-Calculus. In addition it has allowed us to provide tool support for automatically converting AWS IAM policies into Event-Calculus models. The proposed models scale well and even with 100 statements within a policy, the time taken by *DECReasoner* to encode the problem into a SAT problem is 1.1 seconds and solution by relsat solver takes 0.1 seconds. We detail the performance evaluation results in Section 7.

6 Inter-Policy Conflicts

In order to model and verify inter policy conflicts, we group multiple policies in a *PolicySet*. Just as a policy groups multiple statements, a *PolicySet* groups multiple policies. The Event-Calculus models are shown below; due to space limitations we discuss only the instantiated model and corresponding outcome. We model the case where there are two policies, one having only one statement to allow access to EC2 resources (the policy is thus permitted) and the second policy has again only one statement to deny access to EC2 resources (the policy is thus denied). To verify any conflict, we group them in a *PolicySet* as shown below.

PolicySet Model 1 (Instantiated model for grouping policies)

```
policyset EC2PolicySet
PolicySetHasPolicy(EC2PolicySet, AmazonEC2Allow/DenyAccess).
;Goal: Decide if the policySet is permitted/denied/NotApplicable/Invalid
HoldsAt(PolicySetIsPermitted | Denied | Invalid...(policyset),4).
```

The result returned by the *DECReasoner* is shown below. It can be seen that as both the policies evaluated to different decisions at time-point 3, event *InvalidatePolicySet* happens and makes the policy set invalid, as represented by the fluent *PolicySetIsInvalid*.

Solution 3 (PolicySet evaluation by DECReasoner)

```
0
StmtConditionHolds(StmtAllow/StmtDeny).
StmtEffectIsPermit(StmtAllow). Happens(Match(StmtAllow/StmtDeny), 0).
1
+StmtTargetHolds(StmtAllow). +StmtTargetHolds(StmtDeny).
Happens(ApproveStmt(StmtAllow), 1). Happens(DenyStmt(StmtDeny), 1).
2
+StmtIsDenied(StmtDeny). +StmtIsPermitted(StmtAllow).
Happens(Approve/DenyPolicy(AmazonEC2Allow/DenyAccess), 2).
3
+PolicyIsDenied(AmazonEC2DenyAccess).
+PolicyIsPermitted(AmazonEC2AllowAccess).
Happens(InvalidatePolicySet(EC2PolicySet), 3).
4
+PolicySetIsInvalid(EC2PolicySet).
```

The proposed inter-policy conflicts verification approach provides a number of benefits. First the proposed models are intentionally made generic and thus it is easy to model policies, and adding them to a *PolicySet* for the verification, without going into concrete details of Event-Calculus. In addition it has allowed us to provide tool support for automatically converting AWS IAM policies into Event-Calculus models.

7 Implementation and Performance Evaluation

In order to justify the practicality of our approach and to abstract the details of Event-Calculus models, we have developed a Web application[6] to automate the verification process. Our Web application uses AWS access keys and AWS SDK to fetch IAM Users, Groups and their attached policies. The application then allows to first select the IAM Users or Groups and then the Policies that need to be evaluated, Figure 2-A/B. For the verification process, our application automatically generates the Event-Calculus models for the selected AWS policies, invokes the *DECReasoner* and returns the results, Figure 2-C. Space limitations restrict us to discuss the implementation details further.

In order to test the scalability of the proposed approach, we need to scale and verify policies for both intra and inter-policy conflicts. For the Inter-Policy conflicts, we have increased the number of policies assigned to a IAM Group/User and measured the time taken by *DECreasoner* to encode the problem in a SAT problem and the time taken by the *relsat* solver. Instead of merely duplicating a policy to test scalability, we have used the actual AWS Managed IAM policies provided by the AWS. However for the Intra-Policy conflicts, we have manually

[6] Implementation details available at https://members.loria.fr/operrin/files/esocc.txt

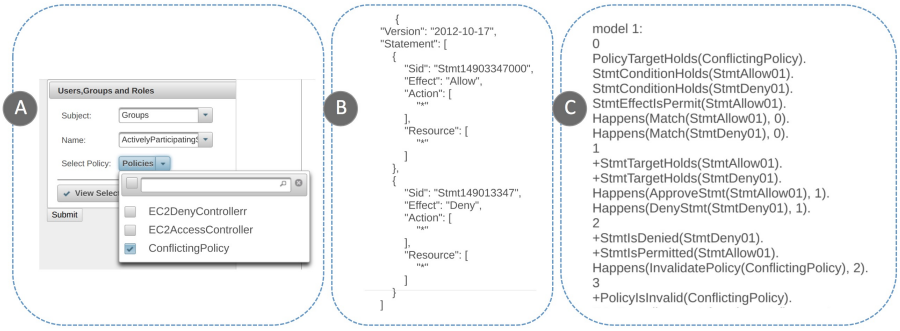

Fig. 2. Automatic conversion from IAM policies to Event-Calculus Models

added statements to a policy as AWS managed policies does not contain a large number of statements as needed to test the scalability of the approach.

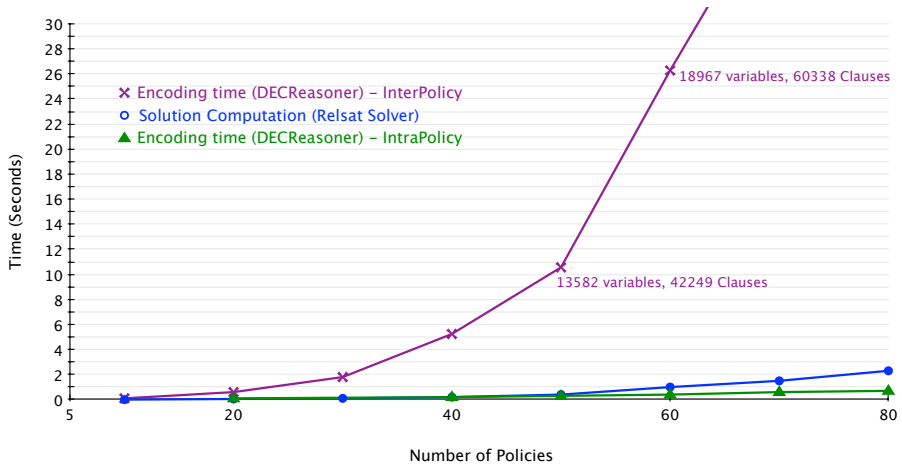

Fig. 3. Performance evaluation results

The performance evaluation test were conducted on a Amazon EC2 m4.2xlarge instance having 8 vCPUs and 32 GiB memory running Ubuntu Server 16.04 LTS. Further, we have used modified and improved *DECreasoner* version as we proposed in [26]. The performance evaluation results are shown in Figure 3, with Y-axis showing the time-taken in seconds while the X-axis showing the increase in the problem size. In general, the solution computation by *relsat* solver is very efficient even with the most complicated models. The Event-Calculus to SAT encoding process in general does not scale well but we have intentionally modeled policies in a way that the axioms do not use a large number of universally

quantified free variables. Thus the SAT encoding also scales reasonably well. The encoding results can be further improved by using incremental encoding or by further improving *DECReasoner* code. For intra policy, the proposed models scale well and even with 100 statements within a policy, the time taken by *DECReasoner* to encode the problem into a SAT problem is 1.1 seconds and solution by relsat solver takes 0.1 seconds.

The performance evaluation results are very encouraging. In order to test the scalability of our approach we intentionally added a number of policies and statements. However, in practice it would be rare to encounter policies with hundreds of statements; for instance the AWS managed (provided) IAM policies have mostly a single statement and in rare cases policies have more then ten statements. Similarly, AWS imposes some limitations on the number of policies attached to a single group (maximum 10 policies can be attached).

8 Conclusion

One of the major challenges in Cloud computing adoption is security and in this paper we address one important security aspect, the Cloud authorization. In contrast to traditional XML (or JSON in case of AWS IAM) based authorization policy specification languages, our approach is formal and based on Event-Calculus, a logical language for specification of and reasoning about events and their effects. The proposed approach can be extended to model other authorization services provided by Cloud providers. For instance, OpenStack provides Role-Based Access Control for networks (Neutron) and user management. Our approach can be used to formally verify and reason about them. We have provided generic Event-Calculus models and provided tool support to automatically convert JSON based IAM policies in Event-Calculus. We have also presented performance evaluation results on actual IAM policies to justify the scalability and practicality of the approach.

References

1. Yu, S., Wang, C., Ren, K., Lou, W.: Achieving secure, scalable, and fine-grained data access control in cloud computing. In: INFOCOM 2010. 534–542
2. Zhu, Y., Huang, D., Hu, C., Wang, X.: From RBAC to ABAC: constructing flexible data access control for cloud storage services. IEEE Trans. Services Computing **8**(4) (2015) 601–616
3. Yang, K., Jia, X.: Expressive, efficient, and revocable data access control for multi-authority cloud storage. IEEE Trans. Parallel Distrib. Syst. **25**(7) (2014) 1735–1744
4. He, H., Li, R., Dong, X., Zhang, Z.: Secure, efficient and fine-grained data access control mechanism for P2P storage cloud. IEEE Trans. Cloud Computing **2**(4) (2014) 471–484
5. Sun, W., Yu, S., Lou, W., Hou, Y.T., Li, H.: Protecting your right: Verifiable attribute-based keyword search with fine-grained owner-enforced search authorization in the cloud. IEEE Trans. Parallel Distrib. Syst. **27**(4) (2016) 1187–1198

6. Park, J.S., Sandhu, R.S., Ahn, G.J.: Role-based access control on the web. ACM Trans. Inf. Syst. Secur. **4**(1) (2001) 37–71
7. Elliott, A., Knight, S.: Role explosion: Acknowledging the problem. In: Proceedings of the 2010 International Conference on Software Engineering Research & Practice, SERP 2010, July 12-15, 2010, Las Vegas, Nevada, USA, 2 Volumes. (2010) 349–355
8. Freudenthal, E., Pesin, T., Port, L., Keenan, E., Karamcheti, V.: drbac: Distributed role-based access control for dynamic coalition environments. In: ICDCS. (2002) 411–420
9. Wu, T., Pei, X., Lu, Y., Chen, C., Gao, L.: A distributed collaborative product design environment based on semantic norm model and role-based access control. J. Network and Computer Applications **36**(6) (2013) 1431–1440
10. Ruan, C., Varadharajan, V.: Dynamic delegation framework for role based access control in distributed data management systems. Distributed and Parallel Databases **32**(2) (2014) 245–269
11. Lee, H.K., Luedemann, H.: lightweight decentralized authorization model for inter-domain collaborations. In: SWS. (2007) 83–89
12. Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to attribute based access control (abac) definition and considerations. NIST Special Publication **800** (2014) 162
13. Bryans, J.: Reasoning about xacml policies using csp. In: SWS. (2005) 28–35
14. Nguyen, T.N., Thi, K.T.L., Dang, A.T., Van, H.D.S., Dang, T.K.: Towards a flexible framework to support a generalized extension of xacml for spatio-temporal rbac model with reasoning ability. In: ICCSA (5). (2013)
15. Kolovski, V., Hendler, J.A., Parsia, B.: Analyzing web access control policies. In: WWW. (2007) 677–686
16. Liang, F., Guo, H., Yi, S., Zhang, X., Ma, S.: An attributes-based access control architecture within large-scale device collaboration systems using xacml. In: Green Communications and Networks. Springer (2012) 1051–1059
17. Tsankov, P., Marinovic, S., Dashti, M.T., Basin, D.A.: Decentralized composite access control. In: POST. (2014)
18. Hardt, D.: The oauth 2.0 authorization framework. (2012)
19. Tootoonchian, A., Saroiu, S., Ganjali, Y., Wolman, A.: Lockr: better privacy for social networks. In: CoNEXT. (2009)
20. Newcombe, C., Rath, T., Zhang, F., Munteanu, B., Brooker, M., Deardeuff, M.: How amazon web services uses formal methods. Commun. ACM **58**(4) (2015)
21. Zahoor, E., Perrin, O., Bouchami, A.: CATT: A cloud based authorization framework with trust and temporal aspects. In: 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing, CollaborateCom 2014, Miami, Florida, USA, October 22-25, 2014. (2014) 285–294
22. Bouchami, A., Perrin, O., Zahoor, E.: Trust-based formal delegation framework for enterprise social networks. In: 2015 IEEE TrustCom/BigDataSE/ISPA, Helsinki, Finland, August 20-22, 2015, Volume 1. (2015) 127–134
23. Montali, M., Maggi, F.M., Chesani, F., Mello, P., Aalst, W.M.P.v.d.: Monitoring business constraints with the event calculus. ACM Trans. Intell. Syst. Technol. **5**(1) (January 2014) 17:1–17:30
24. Kowalski, R.A., Sergot, M.J.: A logic-based calculus of events. New Generation Comput. **4**(1) (1986)
25. Mueller, E.T.: Commonsense Reasoning. Morgan Kaufmann Publishers Inc., CA, USA (2006)
26. Zahoor, E., Perrin, O., Godart, C.: An event-based reasoning approach to web services monitoring. In: ICWS. (2011)