Towards An Attribute Based Constraints Specification Language

Khalid Zaman Bijon, Ram Krishnan and Ravi Sandhu
Institute for Cyber Security
University of Texas at San Antonio

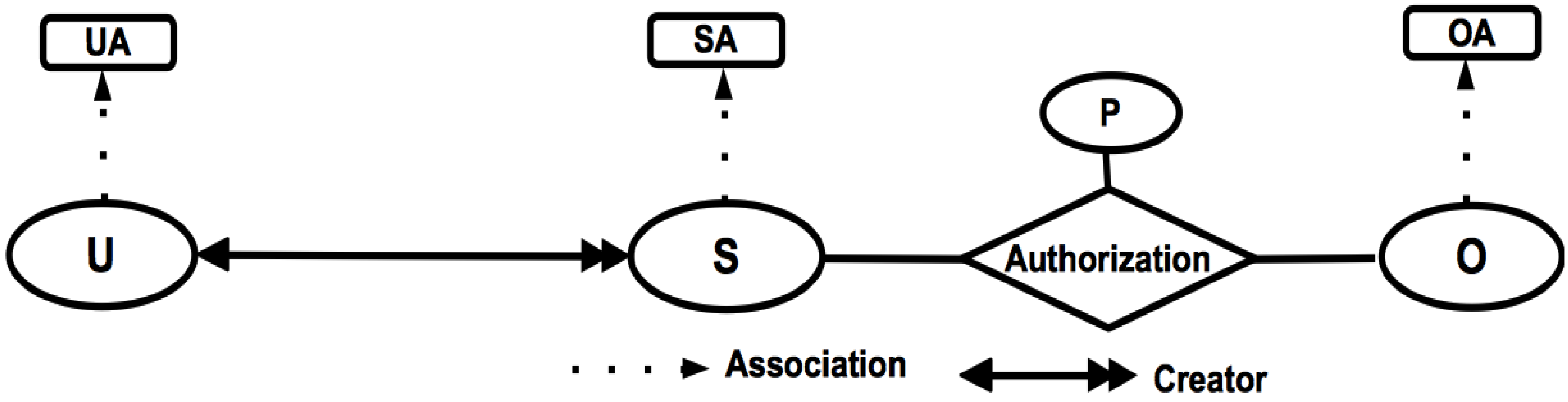
September 11, 2013
2013 ASE/IEEE International Conference on Privacy, Security, Risk and Trust

Attribute Based Access Control (ABAC)

- Emerging as a dominant next generation access control model
  - Policy flexibility and dynamic decision making capability
  - ABAC can express Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role Based Access Control (RBAC)
  - Overcome limitations of DAC, MAC and RBAC

- NIST already released their draft towards a Standard ABAC system (http://csrc.nist.gov/publications/drafts/800-162/sp800_162_draft.pdf)

Overview of an ABAC Model



- User (U), Subject (S) and Object (O) are associate with a set of attributes UA, SA and OA respectively.
- An attribute is a key:value pair. For example, *role* is an attribute and the value of role could be {'president', 'vice-president', 'manager', etc. }
- An attribute can be set-valued or atomic.
  - Clearance vs. Role
- A User needs to create a subject to exercise privileges in the system.
- Each permission is associated with an authorization policy that verifies necessary subject and object attributes for authorization.

Motivation

- ABAC is famous for its policy neutral and dynamic decision making capability
  - Authorization decision of each permission are made by comparing respective attributes of the involved subjects and objects
  - A subject with required attribute can access to an object
- Security policies are necessary to assign attributes to right entities (user, subject, etc.) for avoiding unauthorized access
  - Similar to correct role assignment to users in RBAC
- Proper constraints specification process can configure required security policies of an organization

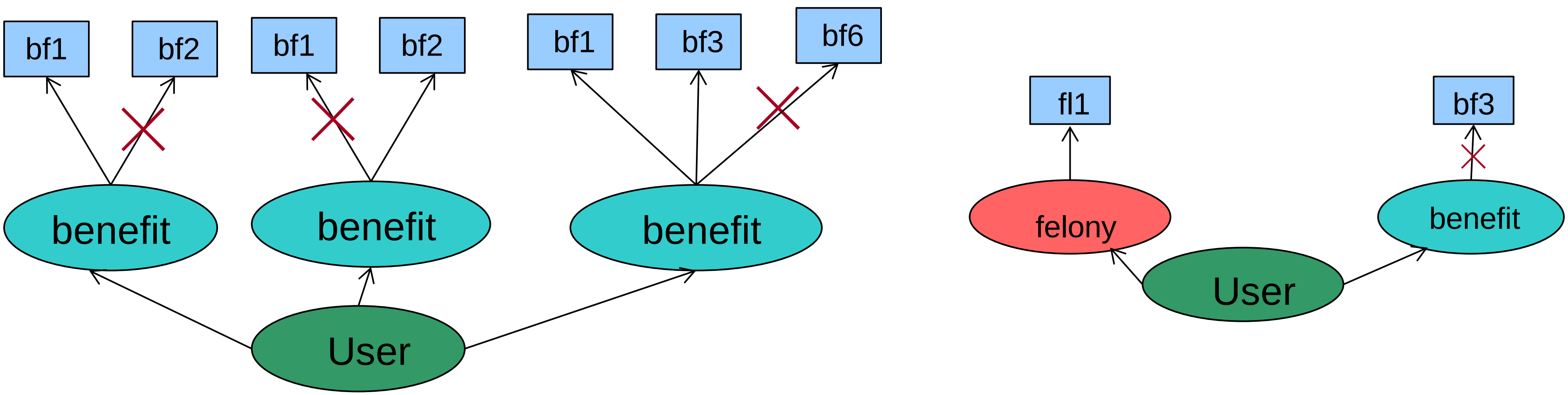
Conducted Research in ABAC

- Attribute Based Access Control Models
  - Focus on ABAC authorization in general, not constraints specification on attribute assignment
  - Lack of proper guideline or process to attribute assignment to entities

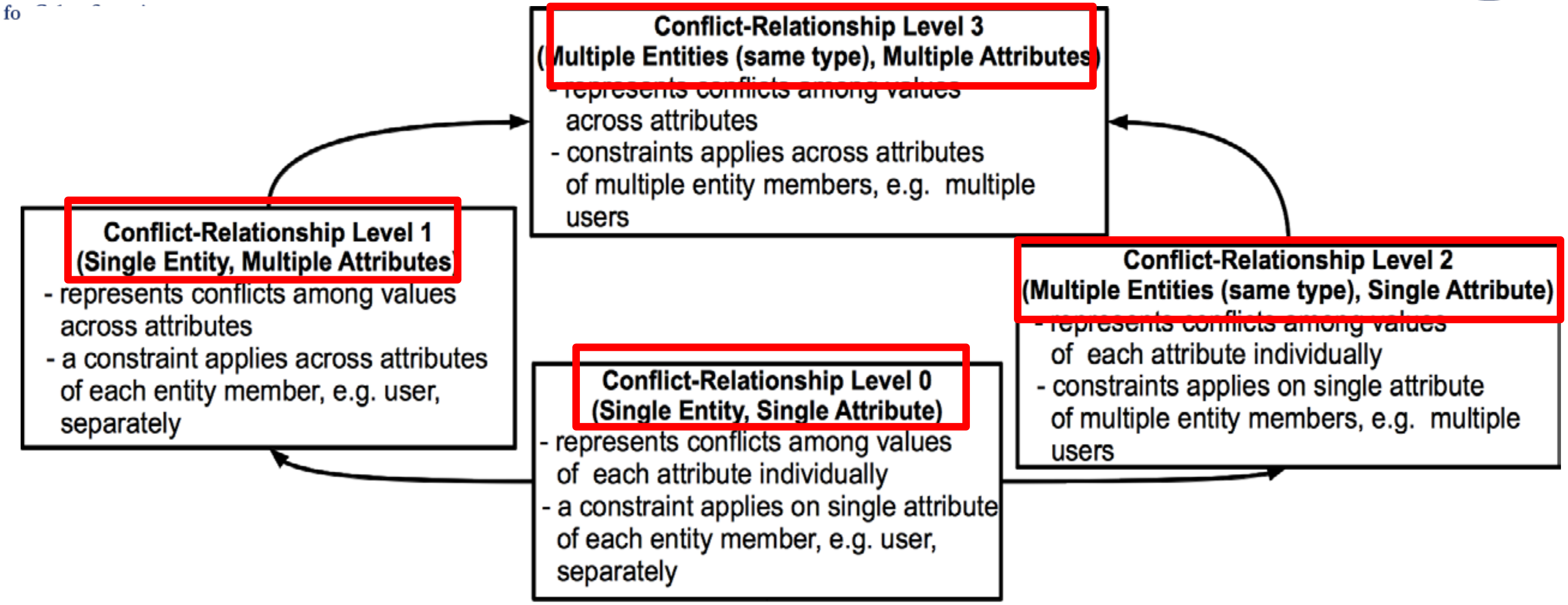
- Attribute Based Encryption
  - Focus on improving encryption process using attributes

- Constraints Specification in Access Control Systems
  - Mainly in RBAC
  - Role Based Constraints Specification Language (RCL-2000)
  - Static and Dynamic Separation of Duty

- Develop an attribute based constraints specification language (ABCL)
  - Identify that attributes preserve different types of conflict-relationship with each other such as mutual exclusion, precondition, etc.
  - A particular conflict-relation restricts an entity to get certain values of an attribute.
    - *Benefit* attribute represents customers' assigned benefits in a Bank
    - A customer cannot get both *benefits* 'bf1' and 'bf2' (mutual exclusion)
    - Cannot get more than 3 benefits from 'bf1', 'bf3' and 'bf6' (cardinality on mutual exclusion)



Attribute Conflict-Relationship Hierarchy



- A constraint can be applied to each entity (one user) separately or across entities (multiple users)
  - *Benefits* 'bf1' cannot be assigned to more than 10 users.
- Hierarchical classification of the attribute conflict-relationships
  - Number of attributes and number of entities are allowed in a conflict relations

Attribute Based Constraints Specification Language (ABCL) (cont.)

- A mechanism to represent different types of such relationships as a set

1. Mutual-Exclusive relation of the *benefit* attribute values (single attribute conflict)

Attribute_Set$_{U,benefit}$ *UMEBenefit*

UMEBenefit={avset1, avset2} where

avset1=({'bf1','bf2'}, 1) and

avset2=({'bf1','bf3','bf4'}, 2)

2. Mutual-Exclusive relation of the *benefit and felony* (cross attribute conflict)

Cross_Attribute_Set$_{U,Aattset,Rattset}$ *UMECFB*

Here, *Aattset*= {felony} and *Rattset*= {benefit}

UMECFB={attfun1} where

attfun1(felony)=(attval, *limit)*

where *attval={'fl1',* 'fl2'} and *limit=1*

attfun1(benefit)=(*attval, limit)*

where *attval={'bf1'}* and *limit=0*

- A grammar in Backus Normal Form (BNF)
  - Declaration of the Attribute_Set and Cross_Attribute_Set
  - Constraint Expression

Declaration of the *Attribute_Set* and *Cross_Attribute_Set*:

<attribute_set_declaration> ::= <atribute_set_type> <set_identifier>
<attribute_set_type> ::= ***Attribute_Set***$_{U,<attname>}$ | ***Attribute_Set***$_{S,<attname>}$ | ***Attribute_Set***$_{O,<attname>}$
<cross_attribute_set_type> ::= ***Cross_Attribute_Set***$_{U,<Aattset>,<Rattset>}$ | ***Cross_Attribute_Set***$_{S,<Aattset>,<Rattset>}$
| ***Cross_Attribute_Set***$_{O,<Aattset>,<Rattset>}$
<*Aattset*> ::= {<attname>, <attname>*}
<*Rattset*> ::= {<attname>, <attname>*}
<set_identifier> ::= <letter> | <set_identifier><letter> | <set_identifier><digit>
<digit> ::= *0|1|2|3|4|5|6|7|8|9*
<letter> ::= *a|b|c|....|x|y|z|A|B|C|...|X|Y|Z*

Constraint Expressions:

<statement> ::= <statement> <connective> <statement> | <expression>
<expression> ::= <token> <atomiccompare> <token> | <token> <atomiccompare> <size>
| <token> <atomiccompare>|<set>| | <token> <atomiccompare> <set> | <token>
<token> ::= <token> <setoperator> <term> | <term> | |<term>|
<term> ::= <function> (<term>) | <attributefun> (<term>) | **OE** (<relationsets>).<item>
| **OE** (<term>) | **OE** (<set>) | **AO** (<term>) | **AO** (<set>) | <attval>
<connective> ::= $\wedge$ | $\Rightarrow$
<setoperator> ::= $\in$ | $\cup$ | $\cap$ | $\notin$
<atomicoperator> ::= $+$ | $<$ | $>$ | $\leq$ | $\geq$ | $\neq$ | $=$
<set> ::= *U*| *S*| *O*
<relationsets> ::= <set_identifier>
<attname> ::= ua_1 | ua_2 | ... | ua_x | sa_1 | sa_2 | ... | sa_y | oa_1 | ... | oa_z
<attval> ::= 'ua_1val_1' | 'ua_1val_2' | ... | 'ua_xval_r' | 'sa_1val_1' | 'sa_1val_2' | ... | 'sa_yval_s' | 'oa_1val_1' | ... | 'oa_zval_t'
<size> ::= ϕ | 1 | ... | N
<item> ::= *limit*| *attval*| **attfun**(<attname>).*limit*| **attfun**(<attname>).*attval*
<attributefun> ::= ua_1 | ua_2 | ... | ua_x | sa_1 | sa_2 | ... | sa_y | oa_1 | ... | oa_z
<function> ::= **SubCreator** | **assignedEntities**$_{U,<attname>}$ | **assignedEntities**$_{S,<attname>}$ | **assignedEntities**$_{O,<attname>}$

Attribute Based Constraints Specification Language (ABCL) (cont.)

- **Examples**

1. A customer cannot get both benefits ‘bf1’ and ‘bf2’

   Expression: |OE(UMEBenefit).attset ∩ benefit(OE(U))| ≤ OE(UMEBenefit).limit

2. If a customer committed felony ‘fl1’, She can not get more than one benefit from ‘bf1’, ‘bf2’ and ‘bf3’

   Expression: OE(UMECFB)(felony).attset ∩ felony(OE(U))| ≥ OE(UMECFB)(felony).limit ⇒ |OE(UMECFB)(benefit).attset ∩ benefit(OE(U))| ≤ OE(UMECFB)(benefit).limit

Use Cases

- ABCL can configure well-known RBAC constraints
  - Role can be considered as a single attribute
  - Can express SSOD and DSOD constraints
  - Just need to declare conflict-relation sets for conflicting roles
- It can configure several security requirements of traditional organization (e.g. banking organization)
  - E.g. Constraints on benefit attribute assignment

Use Cases (cont.)

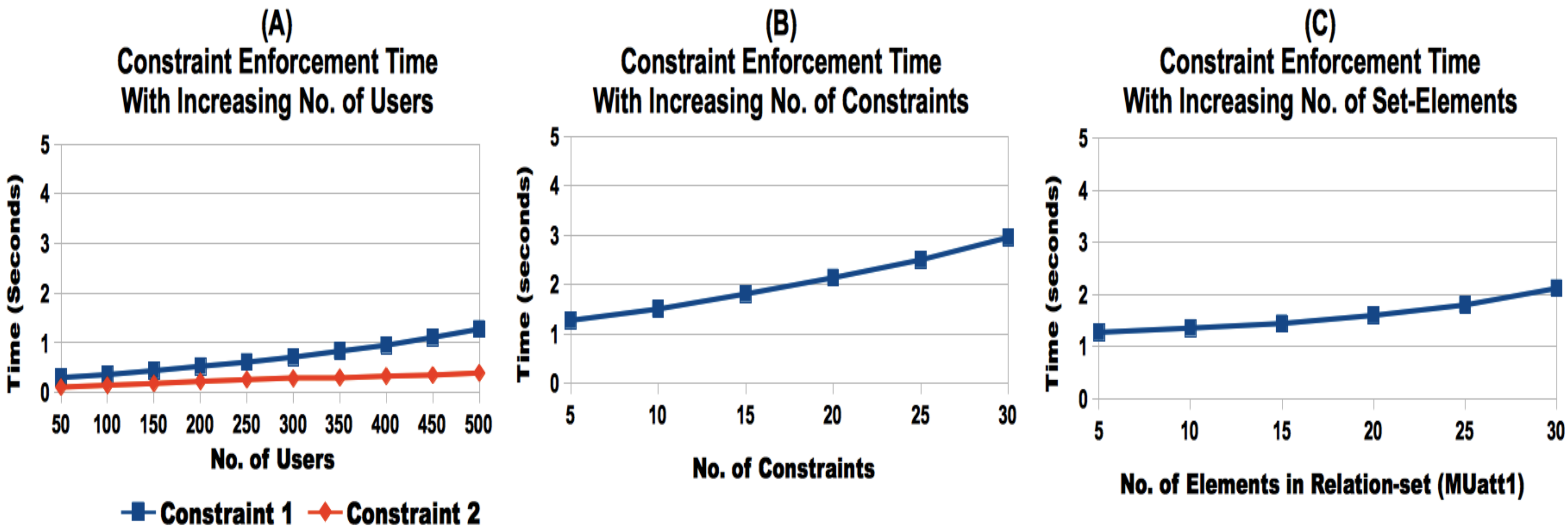
- Security policies for an multi-tenant cloud IaaS
  - Virtual machine (VM) resources management
    - Restricts co-location of VMs from competing tenants (clients)
    - Restrict conflicting workloads from sharing the same memory
    - Other several constraints on resource management
  - Administrative user's privilege management
    - Restricts same admin to gain access on all resources of a client (tenant)
    - Other constraints

ABCL can be implemented as value added service
Provides better service level agreement (SLA) by reducing trust barrier

ABCL Enforcement

- Analyzed Constraints Enforcement complexity
  - Complexity increases in higher level of the relationship hierarchy
- Developed a user attribute assignment algorithm that checks if relevant constraints are satisfied.
- Evaluated the performance of the attribute assignment algorithm

Evaluation

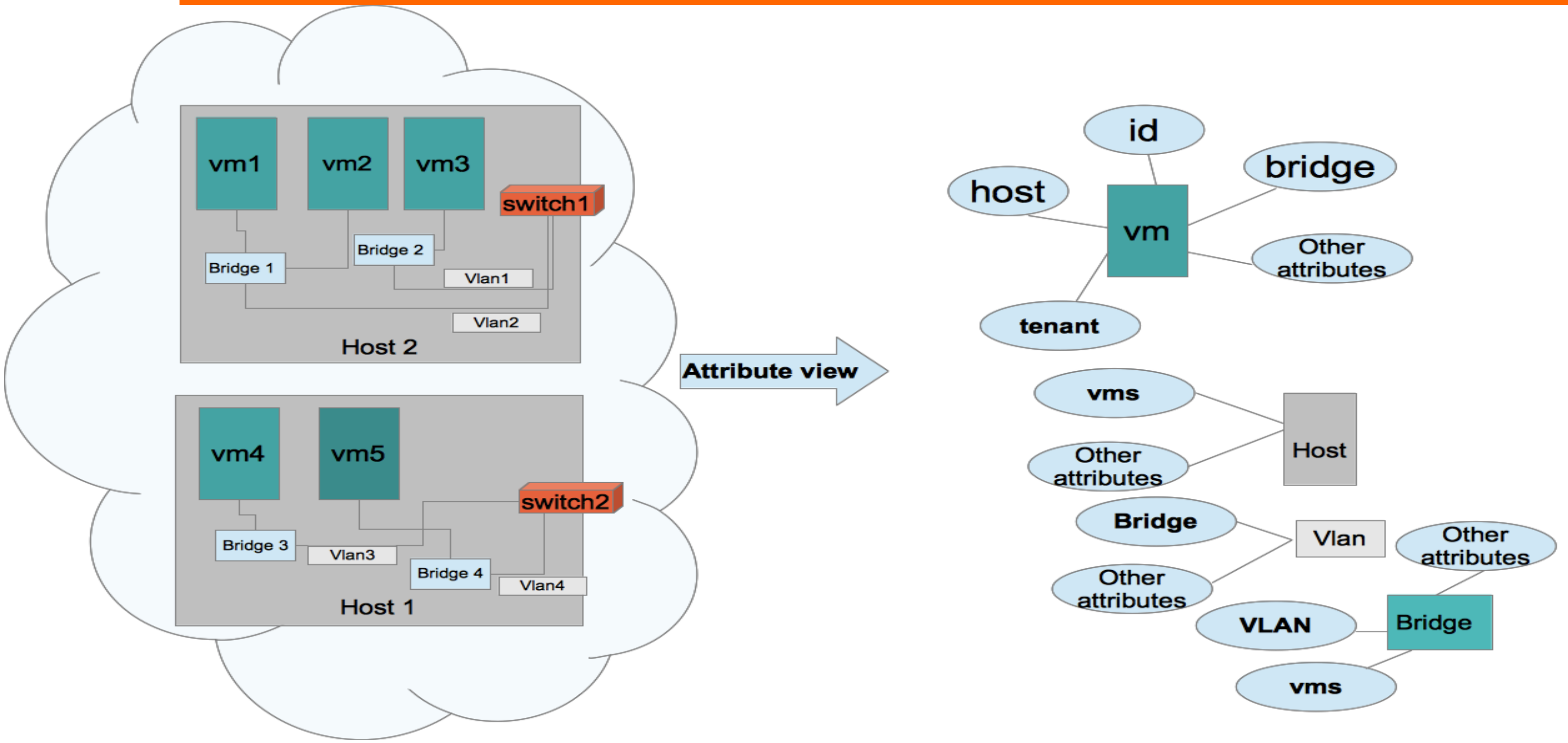


Simulation Scenario:
Constraint #1: each user separately (level 0) , **Constraint #2**: across users (level 2)

Experiment 1: Varying users from 50-500, 2 constraints, 10 elements in relation-set
Experiment 2: 500 users, 5 to 30 different constraints (level 0)
Experiment 3: 500 users, increasing number of set elements (5-30)

Conclusion

A very first investigation on how attributes themselves could be managed based on their intrinsic relationships



- Developing a customized ABCL specification for cloud IaaS in OpenStack
  - Constraint enhanced virtual machine scheduler
- In future, a customized ABCL specification could be developed for resource management in Android Devices

The Institute for Cyber Security

Thank You ☺

Enforcement Complexity

- **Level 0 :** ***O(N ×M ×P)*** where ***N*** is the number of users, ***M*** is the number of elements in respective ***Attribute_Set*** and ***P*** is number of predicates in the expression and their retrieval cost which depends on what data structure has been used.

- **Level 1 :** ***O(N ×(M+O) ×P)*** where ***N*** is the number of users, ***M*** and ***O*** size of ***Attribute_Set*** and ***Cross_Attribute_Set*** respectively, and ***P*** is number of predicates and their retrieval cost

- **Level 2 :** $O(N^2 \times M \times P)$

- **Level 3 :** ***O(N2 ×(M+O) ×P)***