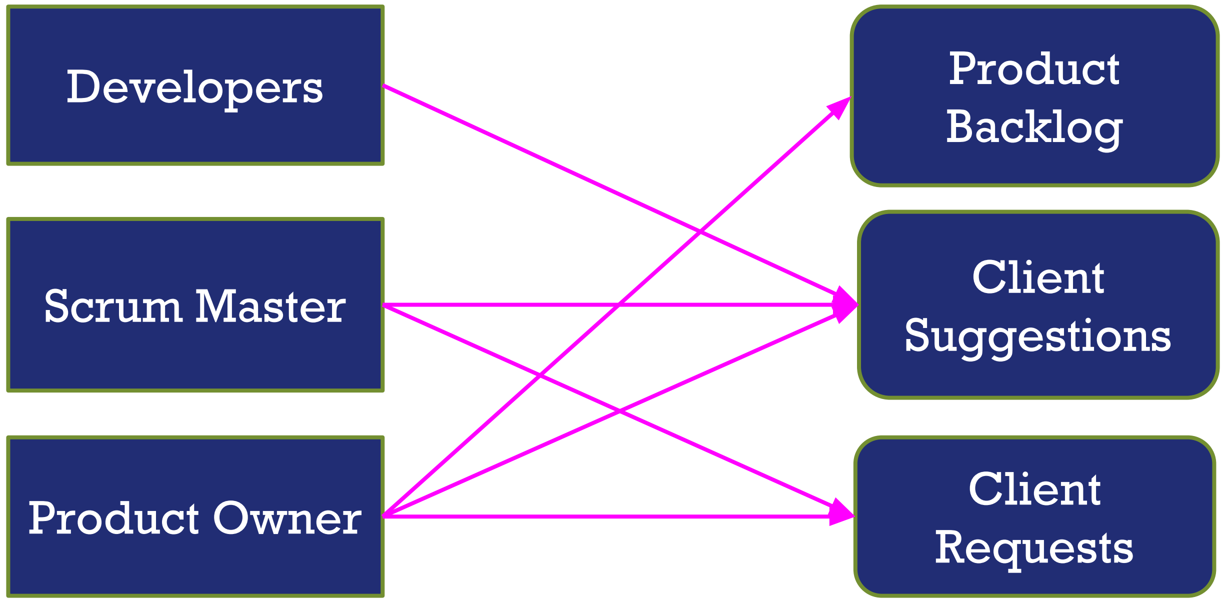
CONSTRAINED POLICY MINING IN ATTRIBUTE BASED ACCESS CONTROL MODELS

TOWARDS ACCESS CONTROL MODELS:

- ❖ With an ever-increasing number of roles and objects, access to documents and resources ought to be regulated.
- ❖ These regulations are implemented in the form of policies, a set of generalized rules that verify/grant/deny access to resources based on the features associated with the resources and requester.

ROLE BASED ACCESS CONTROL MODELS:

- ❖ In this scenario, the access to resources is assigned by using groups of user who have the same authorization. These user groups are a precursor to RBAC.
- ❖ If the role of an individual is reflected in the objects they want to access, the access is granted.



ATTRIBUTE ACCESS CONTROL MODELS:

- In this case, the attributes associated with a user are considered to grant access to the resources they have requested.
- ABAC permissions can be dynamically acquired by virtue of the user attributes.
- While implementing an ABAC model from a RBAC model, all roles associated with a user can be translated as attributes for the user (preserving all role hierarchies).
- But to accomplish ABAC, an exhaustive enumeration of all attributes possible for a user and object must be done a prior.

CONSTRAINED POLICY MINING:

- ❖ A total enumeration of all attributes associated with all user and object together may run into thousands. So, to optimize the response time and space consumed, the weight of each rule must be no more than a specific value.
- ❖ The weight of a rule r is defined as the number of elements in r.
- ❖ For example, ⟨[designation = IC], [evaluationType = Mids], RWE⟩has a weight 3.
- ❖ We denote $TW(P) = \Sigma\, W(r)\ \forall\ r \in P$

TERMINOLOGIES ASSOCIATED WITH CAPM:

- ACM Matrix: This matrix stores the permissions associated (RWE encoded as 3 bit binary numbers ranging from 000 to 111, i.e. 0-7) for the user U_i and object O_j . It's represented as a tuple $\langle U, O, p \rangle$.
- **User-attributes (U_a)**: attributes associated with a user. (department, designation)
- **Object-attributes (O_a)**: attributes associated with an object. (colour, shape, evaluation)
- **UAV(u, a)**: denotes the value of the attribute a, for a given user **v**.
- **OAV(o, a)**: denotes the value of the attribute a, for a given object **o**.
- **Set of User-attribute-name-value pairs (A_u)**
- **Object-attribute-name-value pairs (A_o)**

Algorithm 1: Mine-ABAC-Policy

```
Input: $UP, \mathcal{U}_a, \mathcal{O}_a, UAV, OAV, c$
Output: $Policy$
$i$, $EffectiveRules \leftarrow 0$, $ValidRules[i]$, $RulesCoveredinUP$,
$UniversalUPOSet$, $UOP \leftarrow \phi$
for each $(u, o, p) \in UP$ do
    $UniversalUPOSet \leftarrow UniversalUPOSet \cup \{\langle \mathcal{A}_u, \mathcal{A}_o, p\rangle\}$
end
for each $\langle \mathcal{A}_u, \mathcal{A}_o, p\rangle \in UniversalUPOSet$ do
    $TotalAttrSet \leftarrow \mathcal{A}_u \cup \mathcal{A}_o \cup p$
    $AllPossibleRules \leftarrow r|\ r \in 2^{\mathcal{A}_u \cup \mathcal{A}_o \cup p}$ and $|r| \leq c$
    $ValidRules$[i] $\leftarrow$ r$'$ | r$'$ $\in AllPossibleRules$ and $r'$ covers at
    least one $(u, o, op) \in UP$
    i++
end
for $j \leftarrow 0$ to $i$ do
    $EffectiveRules = EffectiveRules \cup ValidRules[j]$
end
for $j \leftarrow 0$ to $|EffectiveRules|$ do
    $TuplesCoveredinUP[j] \leftarrow$ set of $(u, o, p)$ covered by the
    j$^{th}$ rule in $EffectiveRules$
end
$UOP \leftarrow$ store each $(u, o, p) \in UP$
$TempRules \leftarrow GenMinSetCover(UOP, TuplesCoveredinUP)$
$Policy \leftarrow Merge\text{-}Rules(TempRules)$ return $Policy$
```

Algorithm 2: Merge-Rules

```
Input: $TempRules$
Output: $TempRules$ after merging
$Policy \leftarrow \phi$
for each $(r_1, r_2) \in TempRules$ do
    if $(|r_1 + r_2| \leq c$ and does not uncover any $(u, o, p) \in$
    $UP)$ then
        $r' \leftarrow uc(r_1) \cup uc(r_2) \cup oc(r_1) \cup oc(r_2) \cup op(r_1) \cup$
        $op(r_2)$
        $TempRules \leftarrow TempRules \setminus\{r_1, r_2\} \cup \{r'\}$
    end
end
return $TempRules$
```

ATTRIBUTE HIERARCHY:

- Just as in RBAC, attributes also tend to show a hierarchical model. This hierarchy can be represented as a total order.
- For example, IC $\leq$ Instr $\leq$ PhD TA $\leq$ FDTA.
- This creates a mutual exclusion while creating constructing all rules using A_u U A_o U p.
- Rules which have two or more mutually exclusive attributes associated with them can be deemed semantically incorrect and can be ignored during the construction stage itself. This allows to work with minimum set of rules in the policy for further processing.
- These attribute hierarchies have to be specified a priori (along with the attribute list).

- ❖ **User:** (3 users: U0, U1, U2)
  - ➢ **attributes & values:**

    designation: [IC, Instructor, Professor, FDTA] (IC ≤ Instr ≤ FDTA as well some mutual exclusion)

    department: [CS, EEE, ECE, Civil] (have a mutual exclusion, but no hierarchy)
- ❖ **Object:** (3 objects: O1, O2, O3)
  - ➢ **attributes & values:**

    assignments: [Offline, Online] (have a mutual exclusion, but no hierarchy)

    answer_scripts: [Quiz, Mids, Compre] (Compre ≤ Mids ≤ Quiz)
- ❖ **Permissions:**
  - ➢ [Read, Write, Evaluate]

	O0	O1	O2
U0	1	7	2
U1	7	1	0
U2	1	0	0

	IC	Instructor	Prof	TA	CS	ECE	EEE	Civil
U0	1	1	1	0	0	1	0	0
U1	0	1	0	0	1	0	0	0
U2	0	0	0	1	1	0	0	0

	off	on	quiz	mids	Compre
O0	1	0	1	0	0
O1	0	1	0	0	1
O2	0	1	0	0	1

	IC	Instr	Prof	TA	CS	ECE	EEE	Civil	off	on	quiz	mids	Compre	P
R0	1	0	0	0	0	0	0	0	1	1	0	1	1	2
R1	1	1	1	1	1	1	0	0	0	0	0	1	0	4
R2	1	1	1	0	1	0	0	0	1	1	1	1	1	7
														..
														..

- totalAttrSet ← (A_u U A_o U p): *(*2^{13} primitive rules are created, but some rules can be eliminated as given below during formation itself*)*
- AllPossibleRules ← [rule : W(rule) ≤ c]
- EffectiveRules ← union of all ValidRules

    0. Semantically incorrect/doesn't represent a correct rule
    1. Doesn't represent a correct rule
    2. Represents a proper rule as per ACM (ValidRules)

ALGORITHMS FOR CAPM (My theories)

- For CAPM, I propose the following techniques for size and weight minimization:
  - Naive elimination algorithm
  - Rule elimination algorithm
  - Weight minimization algorithm

Naive Elimination Algorithm

```
for i in range(len(rules)):
    for j in range(len(rules[i])):
        if rules[i][j] == 1:
          rules[i][j] = 0
          if !holds(rule[i]):
              rules[i][j] = 1
    end
end
```

Naive Elimination Algorithm

- In this method, we try to minimize the weight of each rule and by extension, the total weight of all the rules by eliminating those attributes from rules *(setting them to zero)* which do not contribute to the permission level associated to that rule.
- For that, in the given rule, we set each non-zero bit to zero and check if the permission level still holds, i.e. find a similar rule in the ACM with same or higher permission.
- This approach will not reduce the weight of each rule by much, as it requires similar rules to be present in the ACM, which might not always be feasible.

Rule Elimination Algorithm(Step-1)

- A rule is formed by fusing an entry of the UAV and OAV and appending the appropriate permission as per the ACM. Hence, initially we have (N × M) rules. This method allows us to reduce the weight of the whole policy by removing unnecessary/redundant rules:
- Firstly, the rules are divided into clusters or blocks, with each block representing all the available options/values for an attribute. For example, consider the rule 100011110101011000100010 where the first 6 bits represent the values the first attribute (designation) can take, while the next 4 bits represent the different values the second attribute (department) can take (CS, ECE, EEE, Civil).
- Secondly, any duplicate rules are removed.

Rule Elimination Algorithm(Step-2)

- Next, if two rules in the list have same permission, but differ by exactly one bit in the attribute values, we can consider that condition to be a don't-care and remove the rule with the higher weight. This completes the pre-processing.
- In the next step, we try to merge the rules that differ in exactly one block. For this, we examine every pair of rules, and if the necessary condition is satisfied, we can **OR** both the rules to generate the new merged rule and delete the two. This goes well in hand with the hierarchy and mutual exclusion mentioned above. For instance, the *or-ed* rule has the information in the following fashion:
- It is necessary that the two rules to be merged have exactly same permission

Weight Minimization Algorithm

- This is a very rudimentary approach to weight minimization as suggested *Mayank Gautam et al*. In this procedure, we try to discard a few rules that have weight greater than a certain threshold percent.
- Of course, this might lead to removal to rules that represent outliers or multiple merged rules.
- One way to minimize this error margin is to cap the number of times a rule can undergo merging, as multiple merging increase the weight of a rule.
- This cap on the merge count can be fine-tuned by checking for outliers while minimizing the average rule weight and number of tuples that cannot be covered by the policy.

Results and Observations

- The time complexity of the major functions associated with this algorithm are as follows:
  - Pre-processing: $O(N^4 \times M^4)$
  - Merging: $O(N^2 \times M^2 \times \mathit{rule\_size})$
  - Minimizing: $O(N \times M \times \mathit{rule\_size})$

- Overall time complexity: $O(N^4 \times M^4)$

Results and Observations

Sr.no	users	UA	objects	OA	Step-1: total rules	Step-1: total weight of rules	Step-1: avg. weight of rules	Step-2: total rules	Step-2: total weight of rules	Step-2: avg. weight of rules	Time
1	200	27	40	25	7997	168302	21.0456	7997	168302	21.0456	38s
2	350	27	75	25	26233	618023	23.559	26233	618023	23.559	508s
3	450	27	100	25	44960	1020563	22.6994	44960	1020563	22.6994	1365s
4	150	27	35	25	5247	108479	20.6745	5247	108479	20.6745	16s
5	500	27	75	25	37465	786615	20.996	37465	786615	20.996	1156s

Results and Observations

- ❖ The table depicts the number of rules generated after step-1 and step-2 of processing. We observe that a few rules are eliminated during the pre-processing step, but not many rules are eliminated in the merging step, as step-2 requires a one-block variation between two rules, which was not found in our dataset.
- ❖ One side effect that springs up after step-2 is that the average weight of a rule in the policy increases.
- ❖ This is because there is a reduction in the number of rules, but the total weight of all the rules (total number of ones in the policy) almost remain the same.

References

- Mayank Gautam, Sadhana Jha, Shamik Sural, Jaideep Vaidya, Vijayalakshmi Atluri Poster: *Constrained Policy Mining in Attribute Based Access Control (2017)*
- David F. Ferraiolo, D Richard Kuhn: *Role-Based Access Controls (1992)*
- Ed Coyne, Timothy R. Weil: ABAC and RBAC: *Scalable, Flexible and Auditable Access Management (2013)*
- Zhongyuan Xu, Scott D. Stoller: *Algorithms for Mining Meaningful Roles (2012)*
- Zhongyuan Xu, Scott D. Stoller: *Mining Attribute-Based Access Control Policies (2015)*
- David Brossad, Gerry Gebel, Mark Berg: *A Systematic Approach to Implementing ABAC*
- Zhongyuan Xu, Scott D. Stoller: *Mining Attribute-Based Access Control Policies from Logs (2014)*
- Yan Zhu, Dijiang Haung, Chang-Jyun Hu, and Xin Wang: *From RBAC to ABAC: Constructing Flexible Data Access Control for Cloud Storage Services (2015)*
- Tanay Talukdar, Gunjan Batra, Jaideep Vaidya, Vijayalakshmi Atluri, and Shamik Sural: *Efficient bottom-up Mining of Attribute Based Access Control Policies (2017)*

THANK YOU !!