Active Learning of Relationship-Based Access Control Policies

Padmavathi Iyer
University at Albany – SUNY
Albany, New York
riyer2@albany.edu

Amirreza Masoumzadeh
University at Albany – SUNY
Albany, New York
amasoumzadeh@albany.edu

ABSTRACT

Understanding access control policies is essential in understanding the security behavior of systems. However, often times, a complete and accurate specification of the enforced access control policy in a system is not available. In fact, scale and complexity of a system, or unavailability of its source code, may prevent users and even its developers from having access to such accurate specification.

In this paper, we propose a novel, systematic approach for learning access control policies where target systems are treated as black boxes. In particular, we show how we can construct a deterministic finite automaton (DFA) characterizing the relationship-based access control (ReBAC) policy of a system by interacting with its access control engine using minimal number of access requests. Our experiments on realistic application scenarios and their promising results demonstrate the feasibility, scalability and efficiency of our learning approach.

CCS CONCEPTS

• **Security and privacy** → **Access control**; **Authorization**.

KEYWORDS

relationship-based access control, authorization, black box, model learning, active learning

ACM Reference Format:
Padmavathi Iyer and Amirreza Masoumzadeh. 2020. Active Learning of Relationship-Based Access Control Policies. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies (SACMAT '20), June 10–12, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3381991.3395614

1 INTRODUCTION

An access control policy in a system specifies which access requests will be granted/denied. Users need to know and understand the enforced access control policy by a system in order to effectively use the system. System developers also need to ensure that the enforced (implemented) policy meets their security design goals. Unfortunately, most systems today do not come with a clearly documented access control policy; they either completely lack such documentation, or what is documented is inaccurate [30, 26]. This is not necessarily a result of negligence by system developers. Even if they start with a clear policy specification, the implemented system might not conform with the intended specification. It is not trivial to test the correctness of the enforced policy by a typical application today once it has been developed; there will be a daunting access space that needs to be explored. To make the matter worse, many systems today are composed of multiple heterogeneous components that each may impact the end result policy. Even if we know the accurate policy specification for each component, it is very challenging to determine the overall enforced policy as a result of combining those components. We approach this problem by proposing a technique to actively learn the enforced access control policy based on a black-box view of a system. This ensures that we observe and learn the authorization behavior of the system as a whole.

The objective of this paper is closely related to the problem of mining access control policies, which has been extensively explored in the context of role-based access control (RBAC) policies [28, 29, 24, 10], attribute-based access control (ABAC) policies [36, 27, 18, 20, 11, 21], and more recently for relationship-based access control (ReBAC) policies [8, 7, 19, 6]. While mining access control policies is also a learning task it assumes that access control policy information is fully available to learner. For example, most of the above-mentioned work convert low-level policies such as access control lists (ACLs) to a more modern/expressive policy model such as ReBAC. Such an *offline learning* approach is in contrast with our proposed *active learning* strategy that intends to minimize the required knowledge (through observation) of the target system. There are also previous work that intend to learn the policy behavior of a system [22, 26]. However, those work assume that they can exhaustively explore the access space first to generate an authorization log. That problem is essentially reduced to access control mining once such an authorization log has been gathered. Since exhaustive exploration of authorization space is impractical in many real-world systems, those approaches turn to an expensive, manual analysis and reverse engineering of the target system in order to set up assumptions that would limit the considered authorization space. Verifying the correctness of such a manual analysis will be very challenging, if not impossible.

In this paper, we propose a novel approach to infer enforced access control policies. We do not assume having access to source code of the application and assume very minimal knowledge about its data model (See § 2 for details). This approach makes our solution suitable for broader scenarios: for example, when a third party is interested to understand the enforced policy of a system, or when complexities and fast development cycle makes traditional approaches to verifying correctness of access control policies infeasible as discussed earlier. We choose to focus on learning ReBAC policies, which are prevalent in social networking aspects of today's

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
SACMAT '20, June 10–12, 2020, Barcelona, Spain
© 2020 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-7568-9/20/06…$15.00
https://doi.org/10.1145/3381991.3395614

web applications and have been shown to be useful in expressing policies in other domains such as health care as well [16, 12]. In particular, we approach the problem of learning access control policies as a *model learning* problem. Model learning [3, 35] is an active learning paradigm to build a formal representation of a black-box system based on a series of queries submitted to and responded by an abstract entity called a *minimally adequate teacher*.

To the best of our knowledge, this is the first work in the literature to actively infer a formal representation of access control policy enforced by a target system, in a black-box fashion. We summarize our contributions in this work as follows.

- We present a novel and formal active learning approach for learning ReBAC policies from a black-box access control engine, while minimizing the amount of access control queries submitted to observe its behavior (§ 3).
- We propose a learner component (§ 4) to actively infer a deterministic finite automoton (DFA) model of a black-box access control engine, based on our proposed representation of a ReBAC policy as a DFA (§ 2).
- We introduce a mapper component (§ 5) that facilitates such active investigation by abstracting the large access space of target system into relationship patterns that are expressed in ReBAC policies. Rather than static mapping, the mapper component performs inferences that contribute to learning efficiency.
- We propose practical solutions for implementing the equivalence oracle component of our framework (§ 6) that is responsible for evaluating the correctness of a policy DFA inferred by learner.
- Our experiments on two realistic application scenarios, an online social network and an electronic health records system, demonstrates the feasibility, effectiveness and scalability of the proposed solutions (§ 7).

2 DATA MODEL & POLICY MODEL

Table 1 summarizes the notations used in this section and the rest of the paper. In this paper, we focus on applications with rich data models that support *relationship-based access control* (*ReBAC*) policies. In such applications, the data stored in the application can be modeled in the form of a *system graph* [12], where nodes indicate users U and resources R created in the system, and edges represent relationships between them. Let $V = U \cup R$ be the set of entities (users and resources) in a system, L denote the set of edge labels, and E represent the relationships between users and resources in the system. The system graph, which captures the authorization information, is a directed graph $G(V, E)$ where each edge $E \subseteq V \times V \times L$ is labeled by a relationship type in L.

ReBAC policies utilize relationship information contained in the system graph to make access control decisions. The ReBAC access control model was originally developed for the domain of online social networks [17, 16], but has been shown to be applicable to general computing systems like project management and medical records [12, 32]. In this section, we provide a reference model for ReBAC based on the existing literature [16, 12, 32] that encapsulates the necessary authorization-related features in the context of this paper.

Notation	Meaning
G	System graph
U,R,V,E,L	users, resources, vertices, edges, edge labels in G
ϕ	ReBAC rule / relationship pattern
Φ	General notation for set of relationship patterns
Φ_I	ReBAC policy of the SUL
$\Pi_{r,u}$	Set of paths between r and u in G
$\mathcal{R}$	Set of all possible relationship patterns over L
M	General notation for DFA
M_H, M_I	Learner hypothesis DFA, DFA of SUL policy
Q,F,δ,$\mathcal{L}(M)$	States, final states, transitions, language of M
O	Observation table of learner
O_S, O_E	Prefix-closed, suffix-closed set of patterns
O_τ	Function mapping patterns to $\{0, 1\}$
$\cdot$	Concatenation operator
P, $\mathcal{I}$	Mapper mapping table, inference table
P_C	Binary relation of pattern and access request
$\mathcal{I}_\phi$, $\mathcal{I}_{\widehat{\phi}}$, P_ϕ	Set of relationship patterns
$\mathcal{I}_\tau$	Function mapping patterns to {PERMIT, DENY}
C	Possible access requests $\langle u, r \rangle$ (concrete domain)

Table 1: List of Notations

2.1 Authorization Rules

We define a *relationship pattern* ϕ as a sequence of relationship labels $[l_1, l_2, \ldots, l_n]$, where $l_i \in L$. Relationship patterns characterize different arrangements of labeled edges between the entities in the system graph. We assume that the target system determines the *maximum allowable length of relationship patterns*. We denote the domain of relationship patterns by $\mathcal{R}$.

An authorization rule grants accesses based on a relationship pattern. In this paper, we assume that all authorization rules are about granting the same right. This will help simplify discussions by using relationship patterns and authorization rules interchangeably. Extending the rules to consider an extra component that determines applicable right is straightforward.

Finally, the ReBAC policy regulating access control in the target system is a set of authorization rules ϕ_i, and is denoted by Φ_I.

Running Example. We choose a running example in an electronic medical records system that regulates access by doctors, nurses, agents, etc. to patients' medical records. An example authorization rule in this system that allows its doctors to access their patient's medical records could be $[owner,\ doctor]$.

2.2 Authorization Evaluation

We represent an *access request* as tuple $\langle u, r \rangle$, where user $u \in U$ requests to access resource $r \in R$. As discussed in § 2.1, we use an abstract notion of an access right for simplicity; therefore, the right does not need to be specified in an access request.

An access request would be *permitted* only if it matches at least one of the rules in the policy. Let $\Pi_{r,u}$ be the set of paths in G from r to u constrained by the maximum allowable length of relation patterns. Path π matches an authorization rule ϕ if and only if the sequence of edge labels in π matches the sequence of the labels in

ϕ. If a given access request does not match any of the authorization rules in the policy then it will be *denied* access to the resource.

With regards to our running example, suppose Bob issues a request for accessing Alice's record to the electronic application. Access request $\langle Bob, Alice\_Record \rangle$ is permitted if, for example, the following path exists in the system graph that matches the rule given in the example:

$$Alice\_Record \xrightarrow{owner} Alice \xrightarrow{doctor} Bob$$

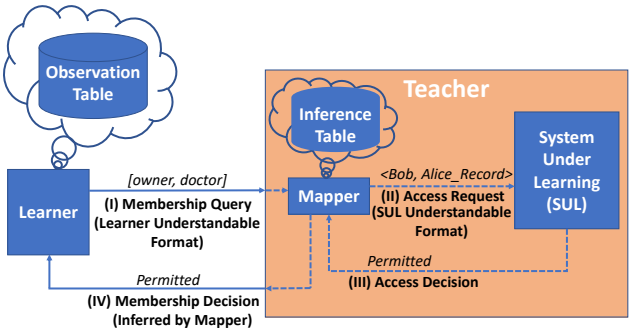
2.3 ReBAC Representation as DFA

Our objective is to infer the access control policy enforced by a system in a black-box manner. We approach this problem by formalizing policy representation as automaton and adopting the framework used for active learning of automata [3]. In the following, we show how a ReBAC policy can be represented as a *deterministic finite automaton (DFA)* and how we bridge the expressiveness gap between the two representations.

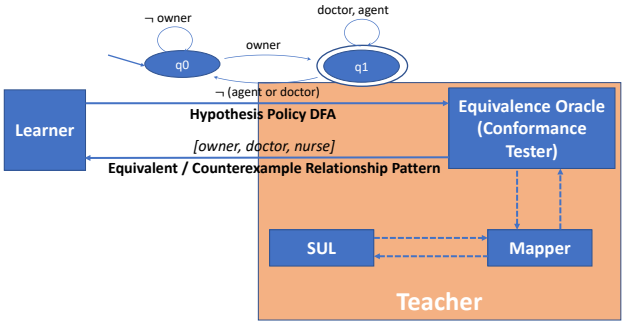
A DFA M is a 5-tuple $\langle Q, L, \delta, q_0, F \rangle$ where Q is a finite set of states, L is the set of alphabets (or relationship labels in our context), $\delta : Q \times L \longrightarrow Q$ is the state transition function, q_0 is the initial state, and $F \subseteq Q$ is the set of accepting (final) states. For brevity of DFA presentation, we use the notation $\neg l$ to imply $\{l' \in L | l' \neq l\}$, i.e., any relationship label but l. Any sequence of transitions $t_1, t_2, \ldots, t_n$ in M, where $q_{i-1} \xrightarrow{t_i} q_i$, forms a string of relationship labels which is equivalent to a relationship pattern. An accepting string in M starts from initial state q_0 and ends in an accepting state $q_n \in F$. The language of M, denoted by $\mathcal{L}(M)$, is the set of all its accepting strings. In our representation of ReBAC as DFA, we consider each accepting string in the DFA as the relationship pattern ϕ_i of an authorization rule. Therefore, $\mathcal{L}(M)$ can represent an authorization policy Φ. Figure 2 shows an example of a ReBAC policy expressed as a DFA. $[owner, doctor]$ and $[owner, doctor, nurse]$ are two of the accepting patterns.

The ReBAC policy model discussed in § 2.1 is not as expressive as a DFA. In particular, our policy does not support relationship patterns with infinite length (which are typically specified using symbols $*$ and $+$ in regular expressions). Therefore, our algorithm learns a constrained DFA model in which any state on an accepting string should not be involved in a cycle. Avoiding cycles on accepting strings ensures that all accepting strings are finite. For example, as shown in Figure 2, the states q_0, q_1, q_2 and q_3 on accepting string $[owner, doctor, nurse]$ are not part of any cycle.

Converting between ReBAC and DFA representations. To further clarify our representation, we briefly discuss how the abovementioned constrained DFA model and ReBAC can be converted to each other. Generating the ReBAC policy corresponding to a DFA is as simple as enumerating its accepting strings as individual rules of the policy. A naive approach for converting a ReBAC policy Φ to its DFA representation would be as follows. Start with a DFA with only an initial state. For any authorization rule $\phi \in \Phi$ in the form of $[l_1, l_2, \ldots, l_n]$, create a set of n states and transitions, where $q_0 \xrightarrow{l_1} q_1 \xrightarrow{l_2} q_2 \xrightarrow{l_3} \ldots \xrightarrow{l_n} q_n$, and mark q_n as a final state. This approach can of course be optimized to construct a minimal DFA.



(a) Interaction Between Learner, Mapper and SUL



(b) Equivalence Oracle for Conformance Testing of Policy DFA

Figure 1: Learning Access Policies from Black-Box Systems

3 OVERVIEW OF PROPOSED APPROACH

Our objective is to extract the underlying authorization rules from a black box system. In addition to inferring the correct policy, we want to minimize the number of access control queries submitted to the system, since such queries are costly and cannot be exhaustively explored in practice. For this reason, we adopt ideas from the *minimally adequate teacher (MAT)* framework [3] for query-based learning of finite automata.

In the MAT framework, a learner has to infer a DFA of a *system under learning* (SUL). In our context, SUL is the *policy decision point* (PDP) that makes authorization decisions for given access requests. So the goal of our learner is to infer a DFA of ReBAC policy enforced in SUL that we refer to as *policy DFA* (§ 2.3). A learner has no prior information about the structure of the policy DFA that it has to infer. However, learner can ask two types of queries, namely *membership* and *equivalence* queries:

- A *membership query* about a string z asks whether z is accepted by SUL. In our context, z is a relationship pattern and the response depends on authorization decision by SUL.
- An *equivalence query* checks whether a hypothesis DFA is correct. In our context, the hypothesis DFA is the policy DFA inferred so far by learner.

Figure 1 shows the architecture of our black-box approach for learning authorization rules. It comprises of four components,

namely *learner*, *mapper*, *equivalence oracle* and *system under learning* (SUL), where the latter three are contained within a larger component called *teacher*. In this figure, the bottom of arrows shows (in bold) the type of query/request submitted and the response received. The top of arrows shows their instances based on our running example (§ 2.1).

The learning procedure works in an iterative manner as follows. Learner submits membership queries about relationship patterns to the teacher. The SUL within teacher accepts only access requests as inputs, so we cannot directly input the relationship patterns from learner to SUL. To tackle this problem, we introduce a mapper component between learner and SUL that accepts relationship patterns from learner and replies authorization decisions by interacting with the SUL (Figure 1a). But a major challenge of this design involves correctly answering the membership queries from learner; errors in the answers to membership queries can impede learner's operation. This could happen because an authorization check by SUL tests if a user can access a resource, but there could be multiple paths between that user and resource in the system graph, out of which only certain paths may be authorized according to the enforced access policies. § 3.2 describes the mapper component in more detail including its strategy for answering membership queries from learner and a crucial challenge in executing that strategy.

After a few rounds of the process described above, based on its observations, learner generates a hypothesis policy DFA and submits it to equivalence oracle for conformance. Equivalence oracle interacts with SUL for obtaining authorization decisions of access requests. If equivalence oracle returns *equivalent*, then the hypothesis is correct and learner has correctly inferred the policy DFA from the system. Otherwise, equivalence oracle replies with a counterexample pattern to the hypothesis, and the learning procedure is resumed after processing the counterexample (Figure 1b). An interesting challenge to note here is that the input to equivalence oracle is a policy DFA whose language comprises of a set of relationship patterns, whereas SUL accepts only access requests. The methodology used by equivalence oracle to verify learner's DFA using SUL is discussed in § 3.3.

Note that in our architecture, the SUL is responsible for performing authorization checks, which is its fundamental purpose as PDP. Execution of membership and equivalence queries is carried out by the learner, mapper and equivalence oracle components. Thus, our learning paradigm can be applied on a legacy implementation of PDP. § 4 describes the working of learner in detail.

3.1 Assumptions and Design Considerations

The followings are our assumptions and considerations regarding the design of the proposed learning architecture.

System graph accessibility We assume that mapper has access to the *system graph* that consists of relationships among users and resources. We note that our focus is inferring the authorization policy imposed partly based on system graph; inferring the system graph itself is out of the scope of this paper.

Immutable system graph An application builds the system graph based on its business logic. Therefore, we cannot assume any fine-grained control over the system graph structure (i.e., directly inserting or deleting nodes/edges).

3.2 Mapper and Infinite User Access Space

As previously mentioned, the SUL in our context is a PDP that can take an access request (i.e., $\langle user, resource \rangle$) as input and output corresponding access decision according to the application's enforced policies. However, as we are interested in learning a ReBAC policy DFA, our learner component is interested in membership queries in the form of relationship patterns. Therefore, we introduce a mapper component as a middleman between learner and SUL. The notion of mapper [2] has been shown to be useful in the MAT framework by dividing the concrete domain of SUL into abstract equivalence classes in a history-dependent manner, thus, reducing the load on learner to infer a finite machine. Our mapper will infer the response to the membership query based on its interactions with SUL, i.e., observing SUL's response to individual access requests. As part of this inference process, mapper produces access requests corresponding to a membership query's relationship pattern and submit them to SUL for evaluation. Moreover, mapper keeps a record of its current inferences about membership queries in an *inference table*.

A significance of our design of mapper is to avoid submitting exhaustive access requests to SUL for evaluation. This design consideration inevitably results in mapper inferring responses to membership queries based on incomplete information. Some of such responses, thus, could be uncertain. This uncertain behavior is limited to some positive responses. Mapper addresses those uncertain responses based on feedback it receives from equivalence oracle. We further explain the details of mapper's inference mechanism and its strategy for updating its inferences in § 5.

Figure 1a shows a sample membership query (relationship pattern $[owner, doctor]$), an access request ($\langle Bob,\ Alice\_Record \rangle$) corresponding to the query pattern that mapper issues to SUL (assuming that $[owner, doctor]$ holds between *Bob* and *Alice's record* in the system graph), and the response it receives from SUL (`PERMIT`). Mapper infers its response to the membership query based on SUL's response and its own inference table, and forwards it to learner.

3.3 Equivalence Oracle and Queries

When learner submits hypothesis policy DFA to equivalence oracle, the latter should find if there is any authorization rule misjudged by learner leading to over-assignment or under-assignment of permissions. If there is no counterexample then equivalence oracle approves the hypothesis generated by learner. Thus, it provides crucial support to learner for comprehending the correct set of access control rules from a black-box system. Figure 1b shows a sample equivalence query (hypothesis DFA) based on our running example (§ 2.1). Suppose the system allows nurse to access the medical records of her doctor's patients. However, this rule is not captured in the hypothesis DFA submitted by learner since the transitions *owner-doctor-nurse* reach the non-accepting state q_0, leading to permission under-assignment. So, equivalence oracle responds with the pattern $[owner, doctor, nurse]$ to learner's hypothesis.

In order to carry out its task of finding counterexamples, equivalence oracle employs system graph information from mapper, along with the ground-truth details about pattern authorizations from mapper's inference table. Besides, equivalence oracle queries the SUL for access decisions to be able to further validate authorized

patterns in the hypothesis. A pleasing characteristic of our architecture is that the counterexamples from equivalence oracle can also be used by mapper for rectifying its wrong inference on pattern authorizations (more details in § 5).

It is indeed challenging to provide equivalence oracle in many real-world learning settings. Equivalence oracle can be approximated using conformance testing. However, to the best of our knowledge, there has been no work in the literature on formal conformance testing of ReBAC policies. To cope with these challenges, in this paper, we present three solutions for implementing equivalence oracle based on various degrees of access space exploration. § 6 explains the operation of equivalence oracle in detail.

4 LEARNER: INFERRING POLICY DFA

In this section, we discuss working of the learner component for actively learning unknown policy DFA in the context of MAT framework as briefly overviewed in § 3.

4.1 Inference Model

Let M_I be the policy DFA, corresponding to ReBAC policy Φ_I, that is enforced by SUL. Learner does not know about the structure of M_I. However, it can submit two types of queries to the teacher in the context of MAT framework:

- *Membership Query*: Learner submits relationship pattern ϕ of its choice to the teacher and obtains whether $\phi \in \mathcal{L}(M_I)$.
- *Equivalence Query*: Learner submits hypothesis DFA M_H and the teacher replies with a *yes* if $\mathcal{L}(M_H) = \mathcal{L}(M_I)$, or otherwise with a counterexample pattern $\phi \in \mathcal{L}(M_I) \,\Delta\, \mathcal{L}(M_H)$ where Δ denotes the symmetric difference operation.

4.2 Observation Table

Observation table is the primary data structure utilized by learner to accommodate knowledge about a finite collection of relationship patterns, distinguishing them as accepted or not accepted. It is defined as triple $\langle O_S, O_E, O_\tau \rangle$. $O_S \subseteq \mathcal{R}$ is a non-empty finite prefix-closed set of relationship patterns. $O_E \subseteq \mathcal{R}$ is a non-empty finite suffix-closed set of patterns. A prefix (suffix)-closed set means that every prefix (suffix) of every member of the set is also a member of the set. O_τ is a finite function $((O_S \cup O_S \,.\, L) \,.\, O_E) \rightarrow \{0, 1\}$ where "." is the concatenation operator. If $O_\tau(\phi)$=1 then the access decision is `PERMIT` and learner expects that ϕ corresponds to a rule in Φ_I, else $O_\tau(\phi)$=0. Learner augments the observation table by submitting membership queries about relationship patterns to the teacher, to which the latter replies with a `PERMIT` or `DENY` decision depending on Φ_I.

An observation table can be visualized as a two-dimensional array whose rows are $(O_S \cup O_S \,.\, L)$ and columns are O_E. The entry for row s and column e is $O_\tau(s.e)$. Let row(ϕ) be the row of pattern ϕ in observation table. An observation table is called *closed* if for every string ϕ' in $(O_S \,.\, L)$ there is a string ϕ in O_S such that row(ϕ')=row(ϕ). An observation table is called *consistent* if whenever $\phi_1, \phi_2 \in O_S$ and row(ϕ_1)=row(ϕ_2) then $\forall l \in L$, row($\phi_1 \,.\, l$)=row($\phi_2 \,.\, l$). The intuition behind observation table is that the rows labeled by elements of O_S are candidates for states of the automaton being constructed, and rows labeled by elements of $(O_S \,.\, L)$ are employed to establish the transition function. Thus, an observation table is eventually used to build a deterministic finite-state automaton.

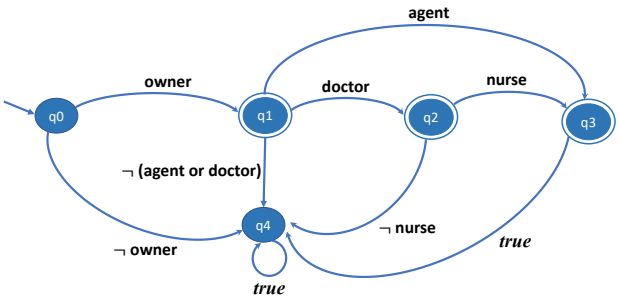


Figure 2: Example DFA for an Electronic Medical Records System

4.3 Learner Methodology

At the beginning, learner populates the observation table with O_S=O_E={σ}, where σ is the empty pattern, by submitting membership queries to the teacher for σ and each $l \in L$. Learner checks the current observation table to see if it is closed and consistent. If not consistent, then learner notes $\phi_1, \phi_2 \in O_S$, $\phi_e \in O_E$ and $l \in L$ such that row(ϕ_1)=row(ϕ_2) but $O_\tau(\phi_1 \,.\, l \,.\, \phi_e) \neq O_\tau(\phi_2 \,.\, l \,.\, \phi_e)$. Then, learner adds the pattern $(l \,.\, \phi_e)$ to O_E and augments O_τ to $(O_S \cup O_S \,.\, L) \,.\, (l \,.\, \phi_e)$ through membership queries for missing elements. If the table is not closed then learner looks for $\phi' \in O_S$ and $l \in L$ such that row($\phi' \,.\, l$) $\neq$ row($\phi \,.\, l$) for all $\phi \in O_S$, and extends O_S by adding the string $(\phi' \,.\, l)$ and augments the observation table by asking membership queries for missing entries.

As soon as the observation table becomes closed and consistent, learner constructs conjecture policy DFA M_H and submits to teacher. If equivalence oracle within teacher confirms the hypothesis and returns *equivalent* then learner successfully terminates with M_H being the final output. On the contrary, if equivalence oracle replies with a counterexample pattern ϕ, then ϕ along with all its prefixes are added to the set O_S (if they don't already exist in O_S) and the function O_τ is expanded to $((O_S \cup O_S \,.\, L) \,.\, O_E)$ by asking membership queries for the missing entries. Learner then repeats the entire procedure on this new observation table $\langle O_S, O_E, O_\tau \rangle$.

In summary, learner issues queries about membership of relationship patterns to the teacher and updates its observation table. Once observation table becomes closed and consistent, learner submits hypothesis policy DFA to teacher. If the teacher replies with a counterexample, then learner updates its observation table to account for the erroneous pattern, and repeats the whole process. Else, learner successfully terminates with right hypothesis.

4.4 Automaton Construction

Let Q be set of states, q_0 be initial state, F be accepting (final) states and δ be transition function, then the DFA M corresponding to a closed, consistent observation table $\langle O_S, O_E, O_\tau \rangle$ and over the alphabet L can be constructed as follows:

Q = {row(ϕ): $\phi \in O_S$},
q_0 = row(σ),

F = {row(ϕ): $\phi \in O_S$ and $O_\tau(\phi)$=1},
δ(row(ϕ), l) = row(ϕ . l).

The transitions in hypothesis policy DFA M_H either comprise of $l \in L$ or $\neg l$ (its negation). If pattern ϕ=[$l_1, l_2, \ldots, l_n$] and $\phi \in \mathcal{L}(M_H)$, then the transition sequence $l_1, l_2, \ldots, l_n$ in M_H from the initial state q_0 results in the accepting state $row(\phi) \in F$. That is, starting from the initial state, traversing the transitions to an accepting state in the hypothesis DFA would yield an authorization rule containing the relationships along the path from the start state to the final state. For example, in Figure 2, the access control rule [$owner, doctor$] is captured by the sequence of transitions $q_0 \xrightarrow{owner} q_1 \xrightarrow{doctor} q_2$ where q_2 is an accepting state. The special transition *true* matches all the elements in the set of relationships.

4.5 Cycle Removal Algorithm

The policy DFA outputted by learner may contain cycles on accepting strings of the automaton, which our ReBAC policies do not support (§ 2.3). Therefore, such cycles need to be removed. To ensure that the policy DFA generated by learner conforms with the specification of our automaton model, we follow a two-step process, namely identifying the cycles and identifying the nodes' reachability to accepting states. This way only those cycles that are on the path to accepting states will be removed.

Following is the pseudo code to detect and prune any cycle present in the accepting strings of automaton M, where the DFA structure is considered as a graph.

(1) Staring from the initial state q_0 of M, traverse the entire automaton in a depth-first manner.
(2) Identify the *back edges* B in the depth-first tree, i.e., those edges (u, v) that connects a vertex u to an ancestor vertex v in the tree, including self-loops. This set B of edges are responsible for cycles in the automaton M.
(3) For every back edge $(u, v) \in B$, determine if v is reachable to any of the accepting states of M. If true, then mark the edge (u, v) for removal.

Once we marked all those edges that are responsible for creating cycles in accepting strings, we want to direct each of the marked edges to a dead state in DFA M. For detecting a dead state, we first determine the connected components of M. Then, the component containing no accepting state is selected and its nodes are marked as dead states. We create a dead state if no such state is found.

4.6 Counterexample Processing

Counterexamples are created when learner's hypothesis fails to correctly capture certain authorization rules. This could be caused in our learner due to two reasons:

- Faults during the inference process on learner's side
- Faults induced as a result of incorrect inference of relationship pattern authorizations by mapper

Suppose that ϕ is the counterexample pattern having length $|\phi|$. For all $i \in \{0, \ldots, |\phi|\}$ let α_i be the access decision produced by processing the first i symbols of ϕ with the hypothesis M_H, and the remaining with the implementation M_I. So, it follows that $\alpha_0 \neq \alpha_{|\phi|}$, which in turn implies that $\exists i_0 \in \{0, \ldots, |\phi| - 1\}$ such that $\alpha_{i_0} \neq \alpha_{i_0+1}$ where such i_0 can be found using a binary search through O(log$|\phi|$) membership queries. Interestingly, the suffix of pattern ϕ starting after the i-th element excluding the first symbol would produce the segment of the ϕ pattern, say p, that causes inconsistency in the observation table.

During counterexample processing, we check if any of the prefixes of the counterexample pattern causes inconsistency in the observation table. If no inconsistency has been caused by any of the prefixes then it indicates that the counterexample ϕ was caused as a result of mapper's incorrect inference. In such a case, we do not process the pattern ϕ as described in § 4.3, rather add a row corresponding to ϕ in the set of rows labeled by (O_S . L). Subsequently, another equivalence query is submitted and the process continues until a counterexample ϕ' is found in which some prefix causes inconsistency, or the observation table $\langle O_S, O_E, O_\tau \rangle$ becomes either not closed or not consistent.

5 MAPPER: ACCESS SPACE ABSTRACTION

This section describes the design and working of the mapper component that mediates between learner and SUL, for learner to infer a DFA. Mapper considers concrete domain of access requests when interacting with SUL, while considering abstract domain of relationship patterns when receiving membership queries from learner.

5.1 Design of the Mapper Component

If U is the set of all users, R is the set of all resources and V is the set of vertices in the system graph $G(V, E)$, then the concrete domain of access requests, $C \subseteq V \times V$, comprises of collection of access requests $\langle u, r \rangle$ where $u, r \in V$. The abstract domain of membership queries (relationship patterns) comprises of all possible relationship patterns $\mathcal{R}$ formed over the relationship labels set L in the system graph G (§ 2). The output of the SUL consists of only two values, `PERMIT` and `DENY`, depending on the decision of access request, and is used to infer membership query responses. So, learner's output domain is the same as that of SUL.

Mapper infers the response to a membership query based on its interactions with SUL and stores its inferences. As part of the inference process, mapper produces an access request associated with the membership query's relationship pattern and submits it to SUL for determining the access decision. Particularly, for a membership query from learner's abstract domain $\phi \in \mathcal{R}$, mapper produces individual access request from SUL's concrete domain $\langle u, r \rangle \in C$. Mapper constructs C based on system graph $G(V, E)$.

Ideally, mapper should return correct responses to learner's membership queries. However, the design of our mapper is such that the access space of SUL is not exhaustively explored. So, it is inevitable that mapper would make inferences about responses to membership queries based on incomplete information. Thus, some of mapper's responses to membership queries could be uncertain. This uncertain behavior is restricted to some positive responses. The inference mechanism used by mapper and its strategy for tackling uncertain responses will be discussed in the rest of this section.

5.1.1 Mapping Table. The *mapping table* is used by mapper to record the associations between relationship patterns and access requests as defined in the system graph. The mapping table is denoted by P, and is specified as a tuple of the form $\langle P_\phi, P_C \rangle$. $P_\phi \subseteq \mathcal{R}$ indicates the set of relationship patterns in system graph G.

$P_C \subseteq P_\phi \times C$ denotes a binary relation that associates each pattern $\phi \in P_\phi$ with its corresponding access requests $\langle u, r\rangle \in C$ such that $\exists \pi \in \Pi_{r,u}$, π matches the pattern ϕ.

For obtaining access requests $\langle u, r\rangle$ corresponding to a given relationship pattern ϕ, we employ the following procedure. Using a graph traversal algorithm, like Breadth-First Search (BFS), we systematically explore the system graph $G(V, E)$ for various relationship patterns defined on the labels set L. For every path $\pi \in \Pi_{r,u}$ between resource $r \in R$ and user $u \in U$ encountered during graph traversal, we record the mapping between ϕ (relationship pattern associated with π) and $\langle u, r\rangle$ into mapper's mapping table P. Since this procedure is repeated for all V nodes in the directed system graph, the time complexity for obtaining the mapping table containing relationship patterns and their associated access requests is $O(V^3)$. For tackling large graphs with several high-degree nodes, our algorithm is provisioned with a user-specified constraint indicating the maximal length of relationship pattern in SUL's enforced policy. So, the graph traversal procedure is length-limited.

5.1.2 Inference Table. The inference table contains mapper's current inferences about responses to learner's membership queries. Formally, the inference table $\mathcal{I}$ can be defined as a tuple $\langle I_\phi, I_{\widehat{\phi}}, I_\tau\rangle$. $\mathcal{I}_\phi \subseteq \mathcal{R}$ is a set of relationship patterns (corresponding to membership queries) for which mapper has inferred a response. $\mathcal{I}_{\widehat{\phi}} \subseteq \mathcal{I}_\phi$ indicates those relationship patterns for which mapper is certain about their responses. Finally, function $\mathcal{I}_\tau : \mathcal{I}_\phi \rightarrow \{\texttt{PERMIT}, \texttt{DENY}\}$ maps each relationship pattern $\phi \in \mathcal{I}_\phi$ to its corresponding inferred response by mapper.

5.2 Working of the Mapper Component

A major design consideration of our mapper is to avoid exhaustive exploration of SUL access space. To this end, mapper infers its responses to membership queries based on minimal interaction with SUL (which may result in uncertain inferences). It also caches its inference in the inference table. The operation of mapper and building its inference table is described as follows.

- At the beginning of the algorithm, the inference table $\mathcal{I}$ is empty, and is augmented as queries are generated by learner.
- Whenever learner submits a pattern query ϕ to mapper, mapper checks its inference table for decision corresponding to that pattern. Specifically, mapper returns $\mathcal{I}_\tau(\phi)$ if $\phi \in \mathcal{I}_{\widehat{\phi}}$.
- If the record does not exist for pattern ϕ in inference table, mapper randomly selects an access request $\langle u, r\rangle$ from its mapping table P where $(\phi, \langle u, r\rangle) \in P_C$, and forwards $\langle u, r\rangle$ to SUL.
- If mapper receives DENY decision from SUL, then none of the relationship patterns Φ between r and u of the submitted access request $\langle u, r\rangle$ can be candidate for authorization rule, and so mapper updates its inference table $\mathcal{I}$ with certain responses. That is, $\forall \phi' \in \Phi$, $\mathcal{I}$ is augmented with $\mathcal{I}_\tau(\phi')$ = DENY and $\mathcal{I}_{\widehat{\phi}} = \mathcal{I}_{\widehat{\phi}} \cup \phi'$.
- For a permitted pattern ϕ, any $\langle u, r\rangle$ where $(\phi, \langle u, r\rangle) \in P_C$ chosen by mapper would result in correct access decision (PERMIT) from the SUL. For a permitted request $\langle u, r\rangle$, if the set of paths between resource r and user u be $\Pi_{r,u} = \{\pi_1, \pi_2, \ldots, \pi_n\}$ that are associated respectively with the patterns $\phi_1, \phi_2, \ldots, \phi_n$ and $\forall i \in \{1, 2, \ldots, n-1\}$ $\mathcal{I}_\tau(\phi_i)$=DENY, then it is intuitive that $\mathcal{I}$ would be updated as $\mathcal{I}_\tau(\phi_n)$=PERMIT and $\mathcal{I}_{\widehat{\phi}} = \mathcal{I}_{\widehat{\phi}} \cup \phi_n$.
- However, for a non-permitted pattern ϕ, mapper could be uncertain about its response since instead of replying DENY to learner, it might forward the PERMIT answer that it received from SUL for some $\langle u, r\rangle$ where $(\phi, \langle u, r\rangle) \in P_C$. This would happen because of the existence of some permitted pattern ϕ', along with the non-permitted ϕ, between r and u in the system graph G. The scenario described here depicts the other case of counterexample occurrence for learner's conjecture whose processing by learner has been already described in § 4.6.
- On mapper's side, such uncertain response about ϕ, where $\phi \in \mathcal{I}_\phi$ but $\phi \notin \mathcal{I}_{\widehat{\phi}}$, is handled by utilizing the feedback received from equivalence oracle during evaluation of learner's conjecture. That is, the inference table is lazily updated with $\mathcal{I}_\tau(\phi)$=DENY record when a counterexample from equivalence oracle arises due to the acceptance of a non-authorized pattern ϕ by learner's hypothesis attributed to mapper's uncertain response. Moreover, $\mathcal{I}$ is updated as $\mathcal{I}_{\widehat{\phi}} = \mathcal{I}_{\widehat{\phi}} \cup \phi$.

6 EQUIVALENCE ORACLE: POLICY TESTING

The correctness of a hypothesis constructed by learner is assessed by submitting an equivalence query to the equivalence oracle component. Equivalence oracle needs to verify the validity of learner's conjecture about access controls enforced in SUL. This is performed using conformance testing. Equivalence oracle employs information from mapper's mapping table and inference table to perform conformance testing. However, note that equivalence oracle does not have direct knowledge about the enforced access control policies since that is a black-box element in the context of this work.

The equivalence oracle component plays a significant role in determining if the conjecture constructed by learner contains any over-assignments or under-assignments of access permissions, and should allow learner to obtain a better comprehension of the access control policy. In this section, we propose three strategies for implementing equivalence oracle each with varying degrees of access space coverage that could be suitable for different scenarios. We note that our implementation of equivalence oracle is the first work in the literature to address conformance testing of ReBAC policies.

We provide details of our general algorithm for conformance testing in the context of our first strategy, i.e., complete access space coverage. However, we note that the first strategy is not efficient for many real-world applications. Our next two strategies limit the access space exploration. Our experimental evaluation shows the effectiveness of such strategies in practice.

6.1 Complete Access Space Coverage

For a given hypothesis DFA M_H, equivalence oracle initially checks the validity of M_H against the ground-truth details about relationship pattern authorizations recorded in mapper's inference table $\mathcal{I}$. In other words, $\forall \phi \in \mathcal{L}(M_H)$, if mapper is certain about the response for pattern ϕ, i.e. $\phi \in \mathcal{I}_{\widehat{\phi}}$, then $\mathcal{I}_\tau(\phi)$ should equal PERMIT. Otherwise, ϕ is returned as counterexample. This evaluation can be applied analogously to patterns not accepted by the hypothesis DFA against the deny records in $\mathcal{I}$.

To further validate relationship patterns in the inferred DFA, equivalence oracle inspects the complete authorization space of SUL. Specifically, it determines the authorizations for every access

request $\langle user, resource \rangle \in C$ by consulting the SUL. Equivalence oracle employs mapper's mapping table P in order to evaluate relationship patterns in hypothesis DFA based on the authorizations of access requests returned by SUL. Subsequently, the following access control considerations are employed by equivalence oracle to determine if there are errors in the constructed hypothesis:

- For a denied request $\langle u, r \rangle$ all paths between resource r and user u based on the system graph should be denied by the hypothesis. Formally, $\forall \phi$ such that $(\phi, \langle u, r \rangle) \in P_C$, it should hold that $\phi \notin \mathcal{L}(M_H)$. Otherwise, ϕ would be a counterexample.
- For a permitted request $\langle u, r \rangle$ at least one of the paths between r and u should be permitted according to learner's hypothesis. Formally, $\exists \phi$ such that $(\phi, \langle u, r \rangle) \in P_C$, $\phi \in \mathcal{L}(M_H)$ should be satisfied. Else, the pattern ϕ would be a counterexample.

Since multiple counterexamples could exist in a given hypothesis DFA, various optimization strategies could be used for selecting the counterexample to be returned to learner. Some of such strategies, which we also used in our experiments, considers pattern length and coverage in access space. Particularly, during counterexample generation, we ensure that smaller patterns are considered before longer counterexample patterns. Besides, if two patterns have the same length then the one with larger number of access requests is selected to ensure that the counterexample pattern with greater coverage in the authorization space is given priority for processing.

Time Complexity Analysis. Suppose the complete user access space in SUL contains $|C|$ records which in the worst case would be $O(|V|^2)$ where V is the vertex set in system graph G. The cost of traversing the DFA would be $O(|Q|^2)$ using a graph traversal algorithm like BFS where Q is the set of states in the DFA. We store the results of exploring the DFA as a mapping between relationship pattern and decision, where the decision specifies if the corresponding pattern ends in an accepting or a non-accepting state.

Assume that we already have the pattern-to-access-request association from mapper's mapping table (§ 5.1.1), and also that the number of different patterns that can exist between any two nodes in the system graph is bounded by b. Since equivalence oracle checks the hypothesis DFA against every access request, in the worst case, the time complexity for executing a complete conformance test would be $O(|Q|^2 + b|V|^2)$. Practically, $|Q| << |V|$ and thus the complexity becomes $O(b|V|^2)$.

6.2 Randomized Approach

The randomized approach selects access requests of users and resources arbitrarily to test the correctness of the access policy hypothesis machine inferred by learner against those selected test cases. Formally, given the complete space of access requests C of U users and R resources such that $C = \{\forall u \in U \ \forall r \in R : \ \langle u, r \rangle\}$, the randomized procedure arbitrarily selects a subset of C which we refer to as $C_{sub} \subseteq C$ and verifies learner's conjecture M_H against this subset C_{sub} of access requests.

The randomized approach for testing the conformity of learner's conjecture is similar to the approach described in the previous section; nevertheless instead of exploring the complete access space, the number of test cases is reduced by a factor of C / C_{sub}, which is more realistic since it is practically infeasible to generate the test cases for the entire user access space. The randomized method would be generally suitable for large applications with huge number of users and resources that are tolerant to small learning inaccuracy due to incomplete test cases.

6.3 Background Knowledge

At times, a domain expert or security administrator may have some domain-specific knowledge regarding the accesses regulated by the target application. For instance, in the scenario of an educational policy, we can expect that instructor of a course should be able to access students' grades for that course. Similarly, in an electronic medical records system, doctors should be given access to their patients' medical records to be able to provide the correct treatment. This kind of domain-related background knowledge about a system's access policy can play an important role during the test case generation phase of the inference process.

Assume that we know that the access control policy should be consistent with rule $\phi = [l_1, l_2, \ldots, l_n]$. Then, we can directly examine the conformance of the learner's DFA M_H instead of consulting the user access space. Specifically, we check if there is a sequence of transitions $t_1, t_2, \ldots, t_n$ in the automaton M_H from the start state q_0 to any accepting state $q_f \in F$ such that $l_i = t_i$. This way, we can avoid generating any test cases between a user and resource whenever they have relationship pattern ϕ between them. Possessing background knowledge about the access controls regulated by a system can significantly aid us in narrowing the space for generating test cases using the randomized approach.

7 IMPLEMENTATION AND EVALUATION

In this section, we discuss our prototype implementation of our access control policy learning framework and evaluate its performance on realistic application scenarios, while providing comparisons with other learning approaches. We perform two kinds of assessments, one for the learning phase and the other for equivalence oracle. The learning phase focuses on interaction between learner, mapper and SUL to evaluate the learning cost in terms of the number of access requests submitted to SUL to infer access rules. To this end, we perform experiments under different learning setups (§ 7.2), and also assess how scalable our approach is with varying sizes of the system graph (§ 7.3). During the learning cost assessment, equivalence oracle is implemented based on complete access space coverage, so our assessment would be independent of equivalence oracle. Finally, we assess the feasibility and performance of implementing randomized equivalence oracle (§ 7.4).

We performed all our experiments on a 64-bit Windows 10 machine using Intel Core i7-6700HQ processor and 12 GB of RAM. The prototype is written in Java v1.8.

7.1 Setup and Configurations

In this section, we describe our sample target applications, the setup of system graph and SUL, and how we acquire the ground truth policies in each case. We assume that relationship patterns in the ground policies will not be more than length 5. So, for all the experiments, we constrained the length of learner queries to 5.

For tackling large graphs with several high-degree nodes, our algorithm is provisioned with a user-specified constraint indicating the maximal length of relationship pattern in learner queries.

Table 2: Learning Cost Comparison Between Our Framework using the Proposed Mapper (Intelligent Mapper), Its Naive Alternative, and An Offline Learning Method

App.	$\|U\|$	$\|R\|$	$\|E\|$	$\|L\|$	Offline Learning	Naive Mapper		Intelligent Mapper	
					#Acc. Req.	$\|Q\|$	*#Acc. Req.*	$\|Q\|$	*#Acc. Req.*
EHR	714	306	25572	8	218484	8	774744	9	37510
Elgg	200	850	21500	4	170000	5	631125	5	6846

7.1.1 Applications Domains. We performed our evaluations on two application domains, namely health-care and social network. We employ a simulated SUL from the health-care domain called *Electronic Health Records* (EHR) that regulates access by doctors, nurses, agents to patients' medical records in a health-care system. From the social network domain, we employ an actual, running application called *Elgg* [14]. Elgg is an open-source social networking software that allows users to add friends, create posts and comment on their friends' posts. The friend relation in Elgg is directional.

7.1.2 Setting Up System Graph and SUL Component.

EHR The system graph was generated by first creating a certain number of user and resource nodes, and then arbitrarily connecting the nodes by taking into account the domain-specific constraints in place. This produced a directed graph connecting users and resources in the institution. The system graph and the access rules (adapted from [16]) are used to simulate the SUL for making authorization decisions for given access requests.

Elgg To create social networking environment comprising of users and their friend network, we utilized random portions of the relationships data from a social network dataset, called soc-Pokec [23]. To generate resources like posts and comments, we mimic actual user interaction with the web application by employing *UI.Vision RPA* tool [34] for user simulation. Given the set of possible atomic user operations, the crawler automatically explores the web application, emulating normal human behavior, by clicking links and filling in details for creating posts and comments.

To establish the SUL component, we used the functions provided by Elgg that deal with authentication to inspect which users have access to different contents. Moreover, we are concerned with inferring the default access control policy provided by the application, rather than the privacy settings available to users.

7.1.3 Ground Truth. For evaluating the correctness of the learned authorization rule set, we utilized the access control policy provided by each target system as the ground truth for comparison. We emphasize that such a ground truth policy is only used for performance evaluation, and not during the learning process.

For *EHR*, we employed the policy specification borrowed from [16]. For Elgg, we referred to the portion of their documentation explaining access control services in the system. Furthermore, we undertook manual inspection to verify conformance of the learned access rules with the enforced policy in the system.

7.2 Learning Cost / Performance Evaluation

For each application, we generate a system graph as discussed earlier and evaluate three different learning setups:

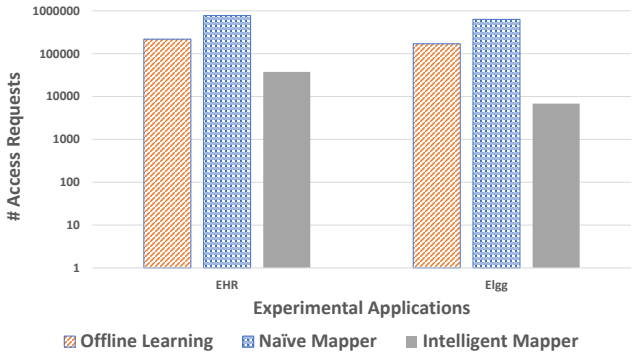


Figure 3: Comparison of # Access Requests to SUL (Log Scale)

Offline Learning (Policy Mining) If no active-learning strategy is used, then in worst case learner would have to explore the complete authorization space of SUL (based on given system graph) to be able to extract the precise system-enforced access policy.

Naive Mapper Whenever learner submits a relationship pattern to naive mapper, the latter simply estimates all the access requests $\langle user, resource \rangle$ corresponding to the pattern, and queries the SUL for access decision on each of those requests. Mapper replies with a `PERMIT` only if all the access requests for the given relationship pattern are granted access according to the authorization policy of the system; otherwise the naive mapper replies `DENY` to learner.

Intelligent Mapper Our proposed mapper as described in § 5 that continually learns and stores access information in its inference table in a history-dependent manner for optimality.

Observations. Table 2 shows our results in each of the above cases. $|U|$, $|R|$, $|E|$, $|L|$ indicate the users, resources, edges and edge labels, respectively, in the system graph. Corresponding to the naive and intelligent mapper cases, $|Q|$ is the number of states in the final policy DFA. "*#Acc. Req.*" is the number of access requests submitted to SUL. We evaluated our learning performance based on semantic similarity (accuracy), privilege over-assignments (false positives) and privilege under-assignments (false negatives). The semantic similarity metric is calculated by taking the ratio of correctly learned access permissions to the whole permission space of ground truth policies. Figure 3 compares the number of access requests submitted to SUL for both applications (in log scale). Following are the observations from our experiments on membership queries.

Least learning cost in intelligent mapper The number of access requests to SUL is lowest for the intelligent mapper, followed by offline learning scenario that requires about 6 times more access requests, and the largest for naive mapper (Figure 3). This is

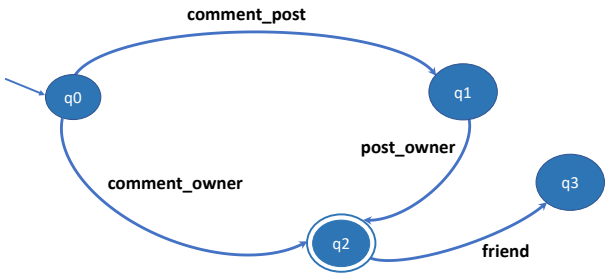


Figure 4: Excerpt of Authorization DFA Inferred from Elgg

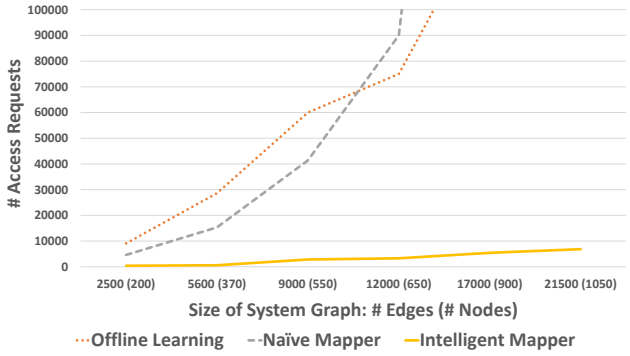


Figure 5: Scalability Evaluation in Elgg

because, compared to the offline approach, active learning overcomes searching through the entire SUL access space by creating hypothesis with just the enough information. Moreover, in contrast to naive mapper, intelligent mapper infers the response to the membership query based on its interactions with SUL and keeps a record of its current inferences about membership queries in its inference table. Additionally, lesser queries in the offline scenario compared to naive mapper is due to repeated querying of access requests by the latter, since we did not cache any query result, that is, its associated access requests and respective access decisions.

More States in intelligent mapper than naive mapper When we inspected the DFA inferred by learner, involving naive mapper, we observed that the learned policy DFA was minimal in terms of number of states and transitions in the automaton. In case of EHR, the policy DFA inferred using intelligent mapper is not minimal unlike its naive counterpart, shown by the difference in the values of $|Q|$ in Table 2. We attribute this to the intelligent mapper's uncertain behavior. Nevertheless, the difference in the DFA size is reasonable compared to the substantial reduction of learning cost in terms of number of access requests issued to SUL.

Accurately learned policy in EHR application For the case of EHR, all the rules were correctly manifested through transitions in the learned automaton. That is, we observed a perfect accuracy score of 1.0, with 0.0 false positives and 0.0 false negatives. In other words, there was no over-assignments or under-assignments of permissions as a result of the learning process.

FN in learned policy of Elgg application Our policy model does not support rules that contain combination of patterns (§ 2). So, accordingly, our learning approach does not consider policy rules that need to satisfy multiple patterns. Experimenting on this real-world social network system rendered interesting observations. Our approach was able to learn rules such as *owners can access their contents including posts and comments*, *owners' friends can access their posts*, and *owners can access comments on their posts*. However, problem arises when extracting rules that are formed by two or more patterns, for example, *users can access comments on others' posts only if they are friends with the comment owner as well as the post owner*. Particularly, the rules containing pattern combinations are denied since our approach functions over one relationship pattern at a time instead of combination of patterns, leading to under-assignment of permissions. The value of semantic similarity was 0.87 and the false negative rate (ratio of under-assignments to permitted accesses) was about 0.63. There was no over-assignments (i.e. 0.0 false positives) in the observed values. The high false negative rate is due to the presence of significantly lesser permitted access requests compared to the denied ones in a practical deny-by-default system.

The abovementioned rule considers a combination of relationship patterns which are [*comment_post*, *post_owner*, *friend*] and [*comment_owner*, *friend*]. Figure 4 depicts part of our inferred policy DFA. Learner has constructed an extra non-accepting state q_3, which is reached by both of those relationship patterns. This implies that its authorization space is identified as distinct compared to other rules. However, as mentioned above, our current algorithms do not support conjunction of relationship patterns which will be in the scope of our future work.

7.3 Scalability Assessment

We examine if our proposed approach is capable of learning authorizations from systems having varying sizes of system graph while submitting least number of access requests to the SUL. For examining the performance of our algorithm with respect to different sizes of the system graph, we obtained random samples of various sizes from the soc-Pokec relationships dataset, and simulated them on the Elgg application. Then, with regards to each of the network sizes, we generate a system graph as explained in § 7.1.2 and executed our learning approach 10 times, and reported the average number of access requests to SUL over the 10 runs.

Observations. Figure 5 demonstrates the number of access requests submitted to SUL in Elgg for learning its policy based on different sizes of system graphs. The horizontal axis depicts the graph growth in terms of # edges as well as # nodes (in brackets), while maintaining certain domain integrity constraints. As the size of system graph increases, both offline and naive mapper graphs increase drastically. However, in the case of intelligent mapper, the number of access requests to SUL is much smaller than those of the other two and also it is growing relatively much slower as the size of the application increases.

7.4 Equivalence Oracle Evaluation

We implemented our equivalence oracle using the randomized strategy (discussed in § 6.2) and evaluated the performance of our

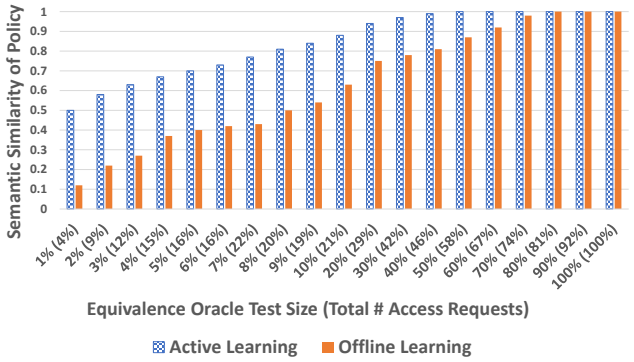


Figure 6: Comparison of Mining Performance Between Active Learning (Proposed Approach) and an Offline Learning Approach [19] for EHR Application.

learning approach depending on the amount of available test cases. Test cases are the randomly selected access requests of users and resources. All reported results are average values over 10 runs.

Figure 6 demonstrates our observations about the accuracy of learned policy for different sizes of equivalence oracle test set for EHR. Here, the horizontal axis represents percentage of user access space considered for equivalence oracle with total access requests submitted to SUL in parentheses. As shown in the figure, with only 30% of access requests provided to equivalence oracle, we achieve an accuracy of about 0.98. Interestingly, with just 1% of access requests, equivalence oracle attains 0.5 accuracy, which manifests the power of our model learning approach for inferring the authorization behavior.

Performance Comparison with Offline Approach. To demonstrate the significance of learning access policies actively, we compare the learning performance of our approach with that of a state-of-the-art offline algorithm from the literature [19] that mines ReBAC policy employing access information provided in the form of access log.

We provided the same access data to the offline algorithm, as was given to our randomized equivalence oracle, based on the total number of queries, including membership and equivalence, issued to SUL in our approach (indicated in parenthesis on the horizontal axis in Figure 6). The results demonstrated in the figure show that, only when more than 80% of the access space is provided, the offline algorithm performs similar to our approach in terms of the accuracy of mined policy. Moreover, the rate at which their approach's accuracy decreases is much faster than that of the proposed algorithm. So, when for 20% of access requests (involving 10% of equivalence oracle test cases) we attain accuracy score of about 0.9, the previous work could reach only around 0.6 accuracy.

8 RELATED WORK

8.1 Mining Access Control Policies

There has been an extensive literature on mining access control policies for different access control models. In role-based access control (RBAC), the idea is to obtain the optimal set of roles from user-permission relation [28, 29]. Subsequently, researchers have looked into the problem of obtaining attribute-based access control (ABAC) policies from role-based policies or Access Control Lists (ACLs) by mining rules based on the attributes associated with users and resources [20, 27, 36, 11, 21]. Furthermore, there has been work on inferring policies expressed as relationships between users and resources (ReBAC) from ACLs or access logs [8, 7, 19, 6]. However, these previous policy mining algorithms assume the provision of complete access request space and that the implementation can be queried regarding access decisions exhaustively. Compared to these previous works on access policy mining, our approach actively infers authorizations through minimal number of queries submitted to the system under learning. To the best of our knowledge, our policy learning algorithm is the first of its kind to employ model learning for inferring policies from black-box applications.

There has been some previous work on mining access control policies from web applications in a black-box fashion. To validate the enforcement of access control policies in web applications, Le et al. present a semi-automated approach to infer those policies even though they may not be clearly specified or documented [22]. Their approach employs the *RandomTree* classifier on a system's user permission space to infer access rules. On the other hand, the focus of this work is to systematically construct the DFA for a black-box system's access control policies in such a way that the number of queries to the implementation is minimized.

8.2 Model Learning

In the recent years, model learning has been applied to numerous practical scenarios spanning over a wide variety of spheres. In the domain of network protocols, researchers have employed combination of model learning and model checking to study different software components including Windows, Linux and FreeBSD implementations of TCP, and their interactions [15]. In addition, there has been work on applying black-box testing to investigate presence of possible flaws in TLS protocol implementations [13].

In the security domain, researchers have proposed a black-box differential testing framework called SFADIFF [4] to automatically detect differences between a set of programs with comparable functionality. Furthermore, model learning has been utilized to analyze regular expression (RE) filters and string sanitizers in a black-box manner [5]. In addition, model learning has been used to deploy active defense mechanism, in a resource-constrained environment, to protect against sophisticated PDF malware [31].

Model learning has also been applied for refactoring legacy embedded software [25, 33] and for reverse engineering smartcards and smartcard readers [1, 9]. To the best of our knowledge, we are the first in the literature to apply model learning for systematically inferring authorization behavior of applications. This is challenging since along with relationship patterns (language of access policy automaton), we deal with overhead of how those patterns correspond to access requests to SUL based on system graph.

9 CONCLUSION AND DISCUSSION

We proposed an active learning methodology for systematically inferring ReBAC policies in a black-box fashion. Our learner component infers the authorizations DFA of the system under learning by employing minimal number of queries to the implementation. Our mapper component works as an intermediary between learner and

the system under learning. Once learner constructs a conjecture of the target automaton, equivalence oracle determines its validity based on conformance testing and returns a counterexample relationship pattern if the testing fails.

We implemented a prototype of our framework and experimented on applications from two domains, electronic health records and online social networks, in order to evaluate the performance of our approach. Our observations demonstrate that our learning algorithm issues significantly lesser number of queries compared to a recently proposed offline learning strategy for ReBAC policies. Moreover, our algorithm is scalable across different sizes of a target system, as manifested by our assessment on the Elgg social network application. Our results show that our proposed randomized approach for implementing equivalence oracle is practical and can achieve significantly better learning accuracy when presented with same amount of test data in comparison to an offline learning approach. We expect that policy background knowledge would further reduce the cost of implementing equivalence oracle, and defer experimenting on real-world scenarios involving such knowledge to future work. Our future work will also include deeper formal analysis of the proposed framework in terms of its learning behavior. We will also explore the effect of other factors on learning performance such as size of the set of relationship labels.

A limitations of our current approach is that our learning process depends on the system graph structure since the mapper and equivalence oracle components depend on it for their respective functionalities. In particular, the system graph structure should be representative of the implemented policy by the system under learning. As future work, we plan to investigate the effect of system graph and formalize the minimum requirements for it to be representative of a policy. Furthermore, extending support for learning more complex and expressive relationship patterns would be of interest. We will explore employing concepts such as product construction of DFAs for this purpose.

REFERENCES

[1] F. Aarts, J. De Ruiter, and E. Poll. Formal models of bank cards for free. In *2013 IEEE Sixth International Conference on Software Testing, Verification and Validation Workshops*, pages 461–468. IEEE, 2013.

[2] F. Aarts, B. Jonsson, J. Uijen, and F. Vaandrager. Generating models of infinite-state communication protocols using regular inference with abstraction. *Formal Methods in System Design*, 46(1):1–41, 2015.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Information and computation*, 75(2):87–106, 1987.

[4] G. Argyros, I. Stais, S. Jana, A. D. Keromytis, and A. Kiayias. Sfadiff: automated evasion attacks and fingerprinting using black-box differential automata learning. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, pages 1690–1701. ACM, 2016.

[5] G. Argyros, I. Stais, A. Kiayias, and A. D. Keromytis. Back in black: towards formal, black box analysis of sanitizers and filters. In *2016 IEEE Symposium on Security and Privacy (SP)*, pages 91–109. IEEE, 2016.

[6] T. Bui, S. D. Stoller, and H. Le. Efficient and extensible policy mining for relationship-based access control. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, pages 161–172, 2019.

[7] T. Bui, S. D. Stoller, and J. Li. Greedy and evolutionary algorithms for mining relationship-based access control policies. *Computers & Security*, 80:317–333, 2019.

[8] T. Bui, S. D. Stoller, and J. Li. Mining relationship-based access control policies. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, pages 239–246. ACM, 2017.

[9] G. Chalupar, S. Peherstorfer, E. Poll, and J. De Ruiter. Automated reverse engineering using lego®. In *8th {USENIX} Workshop on Offensive Technologies ({WOOT} 14)*, 2014.

[10] A. Colantonio, R. Di Pietro, and A. Ocello. A Cost-driven Approach to Role Engineering. In *Proceedings of the 2008 ACM Symposium on Applied Computing*, SAC '08, pages 2129–2136, New York, NY, USA. ACM, 2008.

[11] C. Cotrini, T. Weghorn, and D. Basin. Mining abac rules from sparse logs. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 31–46. IEEE, 2018.

[12] J. Crampton and J. Sellwood. Path conditions and principal matching: a new approach to access control. In *Proceedings of the 19th ACM symposium on Access control models and technologies*, pages 187–198. ACM, 2014.

[13] J. De Ruiter and E. Poll. Protocol state fuzzing of TLS implementations. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 193–206, 2015.

[14] Elgg. https://elgg.org/, 2004.

[15] P. Fiterău-Broştean, R. Janssen, and F. Vaandrager. Combining model learning and model checking to analyze tcp implementations. In *International Conference on Computer Aided Verification*, pages 454–471. Springer, 2016.

[16] P. W. Fong. Relationship-based access control: protection model and policy language. In *Proceedings of the first ACM conference on Data and application security and privacy*, pages 191–202. ACM, 2011.

[17] P. W. Fong and I. Siahaan. Relationship-based access control policies and their policy languages. In *Proceedings of the 16th ACM symposium on Access control models and technologies*, pages 51–60. ACM, 2011.

[18] M. Gautam, S. Jha, S. Sural, J. Vaidya, and V. Atluri. Poster: constrained policy mining in attribute based access control. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies*, pages 121–123. ACM, 2017.

[19] P. Iyer and A. Masoumzadeh. Generalized mining of relationship-based access control policies in evolving systems. In *Proceedings of the 24th ACM Symposium on Access Control Models and Technologies*, pages 135–140. ACM, 2019.

[20] P. Iyer and A. Masoumzadeh. Mining positive and negative attribute-based access control policy rules. In *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*, pages 161–172. ACM, 2018.

[21] L. Karimi and J. Joshi. An unsupervised learning based approach for mining attribute based access control policies. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 1427–1436. IEEE, 2018.

[22] H. T. Le, C. D. Nguyen, L. Briand, and B. Hourte. Automated inference of access control policies for web applications. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, pages 27–37. ACM, 2015.

[23] J. Leskovec and A. Krevl. SNAP Datasets: Stanford large network dataset collection. http://snap.stanford.edu/data, June 2014.

[24] H. Lu, J. Vaidya, and V. Atluri. Optimal Boolean Matrix Decomposition: Application to Role Engineering. In *2008 IEEE 24th International Conference on Data Engineering*, pages 297–306, Apr. 2008.

[25] T. Margaria, O. Niese, H. Raffelt, and B. Steffen. Efficient test-based model generation for legacy reactive systems. In *Proceedings. Ninth IEEE International High-Level Design Validation and Test Workshop (IEEE Cat. No. 04EX940)*, pages 95–100. IEEE, 2004.

[26] A. Masoumzadeh. Inferring unknown privacy control policies in a social networking system. In *Proceedings of the 14th ACM Workshop on Privacy in the Electronic Society*, pages 21–25. ACM, 2015.

[27] E. Medvet, A. Bartoli, B. Carminati, and E. Ferrari. Evolutionary inference of attribute-based access control policies. In *International Conference on Evolutionary Multi-Criterion Optimization*, pages 351–365. Springer, 2015.

[28] B. Mitra, S. Sural, J. Vaidya, and V. Atluri. A Survey of Role Mining. *ACM Comput. Surv.*, 48(4):50:1–50:37, Feb. 2016. ISSN: 0360-0300.

[29] I. Molloy, N. Li, Y. A. Qi, J. Lobo, and L. Dickens. Mining roles with noisy data. In *Proceedings of the 15th ACM symposium on Access control models and technologies*, pages 45–54. ACM, 2010.

[30] T. OWASP. 10: ten most critical web application security risks, 2013.

[31] Z. Perumal and K. Veeramachaneni. Towards building active defense systems for software applications. In *International Symposium on Cyber Security Cryptography and Machine Learning*, pages 144–161. Springer, 2018.

[32] S. Z. R. Rizvi, P. W. Fong, J. Crampton, and J. Sellwood. Relationship-based access control for an open-source medical records system. In *Proceedings of the 20th ACM Symposium on Access Control Models and Technologies*, pages 113–124. ACM, 2015.

[33] M. Schuts, J. Hooman, and F. Vaandrager. Refactoring of legacy software using model learning and equivalence checking: an industrial experience report. In *International Conference on Integrated Formal Methods*, pages 311–325. Springer, 2016.

[34] Ui.vision rpa. https://ui.vision/rpa, 2016.

[35] F. Vaandrager. Model learning. *Commun. ACM*, 60(2):86–95, Jan. 2017. ISSN: 0001-0782.

[36] Z. Xu and S. D. Stoller. Mining attribute-based access control policies. *IEEE Transactions on Dependable and Secure Computing*, 12(5):533–545, 2014.