The Next 700 Policy Miners: A Universal Method for Building Policy Miners

Carlos Cotrini
Department of computer science
ETH Zürich
ccarlos@inf.ethz.ch

Luca Corinzia
Department of computer science
ETH Zürich
luca.corinzia@inf.ethz.ch

Thilo Weghorn
Department of computer science
ETH Zürich
thilo.weghorn@inf.ethz.ch

David Basin
Department of computer science
ETH Zürich
basin@inf.ethz.ch

ABSTRACT

A myriad of access control policy languages have been and continue to be proposed. The design of policy miners for each such language is a challenging task that has required specialized machine learning and combinatorial algorithms. We present an alternative method, *universal access control policy mining* (Unicorn). We show how this method streamlines the design of policy miners for a wide variety of policy languages including ABAC, RBAC, RBAC with user-attribute constraints, RBAC with spatio-temporal constraints, and an expressive fragment of XACML. For the latter two, there were no known policy miners until now.

To design a policy miner using Unicorn, one needs a policy language and a metric quantifying how well a policy fits an assignment of permissions to users. From these, one builds the policy miner as a search algorithm that computes a policy that best fits the given permission assignment. We experimentally evaluate the policy miners built with Unicorn on logs from Amazon and access control matrices from other companies. Despite the genericity of our method, our policy miners are competitive with and sometimes even better than specialized state-of-the-art policy miners. The true positive rates of policies we mined differ by only 5% from the policies mined by the state of the art and the false positive rates are always below 5%. In the case of ABAC, it even outperforms the state of the art.

CCS CONCEPTS

• **Computing methodologies → Supervised learning by classification**; • **Security and privacy → Access control**.

KEYWORDS

access control; policy mining; security policies; machine learning

ACM Reference Format:
Carlos Cotrini, Luca Corinzia, Thilo Weghorn, and David Basin. 2019. The Next 700 Policy Miners: A Universal Method for Building Policy Miners. In *2019 ACM SIGSAC Conference on Computer and Communications Security (CCS '19), November 11–15, 2019, London, United Kingdom.* ACM, New York, NY, USA, 18 pages. https://doi.org/10.1145/3319535.3354196

1 INTRODUCTION

1.1 Motivation and research problem

Numerous access control policy languages have been proposed over the last decades, e.g., RBAC (Role-Based Access Control) [26], ABAC (Attribute-Based Access Control) [37], XACML (eXtended Access-Control Markup Language) [34], and new proposals are continually being developed, e.g., [7, 10, 16, 57, 78]. To facilitate the policy specification and maintenance process, policy miners have been proposed, e.g., [12, 18, 31, 33, 43, 52, 56, 77]. These are algorithms that receive an assignment of permissions to users and output a policy that grants permissions to users that match as closely as possible the given assignment.

Designing a policy miner is challenging and requires sophisticated combinatorial or machine-learning techniques. Moreover, policy miners are tailor-made for the specific policy language they were designed for and they are inflexible in that any modification to the miner's requirements necessitates its redesign and reimplementation. For example, miners that mine RBAC policies from access control matrices [31] are substantially different from those that mine RBAC policies from access logs [56]. As evidence for the difficulty of this task, despite extensive work in policy mining, no miner exists for XACML [34], which is a well-known, standardized language.

Any organization that wishes to benefit from policy mining faces the challenge of designing a policy miner that fits its own policy language and its own requirements. This problem, which we examine in Section 3, is summarized with the following question: *is there a more general and more practical method to design policy miners?*

1.2 Contribution

We propose a radical shift in the way policy miners are built. Rather than designing specialized mining algorithms, one per policy language, we propose Unicorn, *a universal method for building policy miners.* Using this method, the designers of policy miners no longer must be experts in machine learning or combinatorial optimization

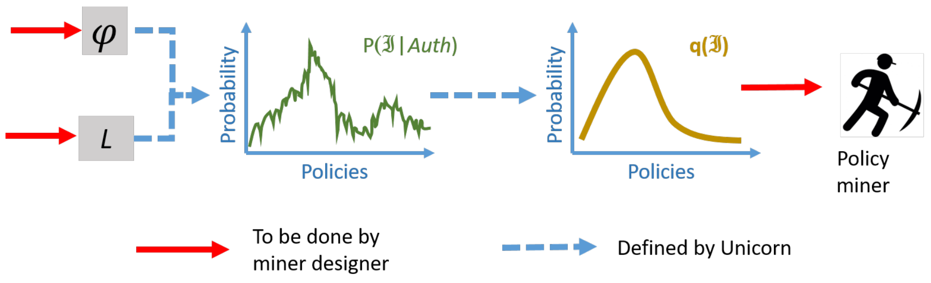

Figure 1: Workflow for designing a policy miner using Unicorn.

to design effective policy miners. Our method gives a step-by-step procedure to build a policy miner from just the *policy language* and an *objective function* that measures how well a policy fits an assignment of permissions to users.

Let Γ be a policy language. We sketch below and in Figure 1 the workflow for designing a policy miner for Γ using Unicorn.

Policy language and objective function (Sections 4 and 5). The miner designer specifies a *template formula for* Γ in a fragment $\mathcal{L}$ of first-order logic. Template formulas are explained in Section 4. The designer also specifies an objective function L that measures how well a policy fits a permission assignment.

Probability distribution (Section 5). From φ and L, we define a probability distribution $\mathbb{P}$ on policies, conditioned on permission assignments. A permission assignment is a relation between the set of users and the set of permissions. The policy miner is a program that receives as input a permission assignment *Auth* and aims to compute the most likely policy conditioned on *Auth*; that is, the policy $\mathfrak{I}$ that maximizes $\mathbb{P}(\mathfrak{I} \mid Auth)$.

Approximation (Section 6). Computing $\max_{\mathfrak{I}} \mathbb{P}(\mathfrak{I} \mid Auth)$ takes time exponential in the size of *Auth* and $\mathfrak{I}$, encoded as strings. Moreover, the function $\mathbb{P}(\mathfrak{I} \mid Auth)$ has many local maxima. Hence, we use *deterministic annealing* and *mean-field approximation* [9, 11, 61, 62] to derive an iterative procedure that computes a distribution q on policies that approximates $\mathbb{P}(\mathfrak{I} \mid Auth)$. Computing $\arg\max_{\mathfrak{I}} q(\mathfrak{I})$ takes time polynomial in the size of *Auth* and $\mathfrak{I}$.

Implementation (Section 7). The policy miner is a procedure that computes and maximizes q. One need not understand mean-field approximations or deterministic annealing to implement the policy miner. We provide a set of rewriting rules and pseudocode that guide step by step q's computation and maximization (see Algorithm 1 and Lemma 2).

In summary, designing a policy miner for a policy language previously required expertise in machine learning and combinatorial algorithms. Unicorn reduces this to the task of specifying a template formula and implementing q's maximization. We illustrate how specifying template formulas requires only the background in first-order logic provided in this paper and how it amounts to just formalizing the language's semantics in first order logic, a task that is substantially simpler than designing a machine-learning or a combinatorial algorithm.

1.3 Applications and evaluation

Using Unicorn, we have built miners for different policy languages like RBAC, ABAC, and RBAC with user attributes. Furthermore, we have built policy miners for RBAC with spatio-temporal constraints and an expressive fragment of XACML, for which no miner existed before. We present them in Sections 8 and 9 and in the appendix.

In Section 10, we conduct an extensive experimental evaluation using datasets from all publicly available real-world case studies on policy mining. We compare the miners we built with state-of-the-art miners on both real-world and synthetic datasets. The true positive rates of the policies mined by our miners are within 5% of the true positive rates of the policies mined by the state of the art. For policy languages like XACML or RBAC with spatio-temporal constraints, the true positive rates are above 75% in all cases and above 80% in most of them. The false positive rates are always below 5%. For ABAC policies, we mine policies with a substantially lower complexity and higher precision than those mined by the state of the art. This demonstrates that with Unicorn we can build a wide variety of policy miners, including new ones, that are competitive with or even better than the state of the art.

Unicorn's effectiveness follows from the wide applicability of deterministic annealing (DA). This technique has been applied to different optimization problems like the traveling salesman problem [61], clustering [62], and image segmentation [36]. DA can also be applied to policy mining. However, in our case, computing the distribution $\mathbb{P}(\mathfrak{I} \mid Auth)$ required by DA is intractable. Hence, we use mean-field approximation (MFA) to compute a distribution q that approximates $\mathbb{P}$. This distribution q is much easier to compute. Moreover, our approach of DA with MFA turns out to generalize to a wide variety of policy languages.

We examine related work and draw conclusions in Sections 11 and 12. For details on deterministic annealing and mean-field approximation, we refer to the literature [9, 11, 61, 62].

2 PRELIMINARIES

2.1 Policy mining

Organizations define *organizational policies* that specify which permissions each user in the organization has. Such policies are usually described in a high-level language. To be machine enforceable, policy administrators must specify this policy as an *(access control) policy* in a machine-readable format. This policy assigns *permissions* to *users* and is formalized in a *policy language*. The policy is then enforced by mechanisms that intercept each *request* (a pair consisting of a user and a permission) and check whether it is authorized.

Organizations are highly dynamic. New users come and existing users may go. Moreover, groups of users may be transferred to other organizational units. Such changes induce changes in the access control policy, which are usually manually implemented, giving rise to the following problems. First, the policy may become convoluted and policy administrators no longer have an overview on who is authorized to do what. Second, policy administrators may have granted to users more permissions than needed to do their jobs. This makes the organization vulnerable to abuse by its own users, who may exploit the additional permissions and harm the organization.

To address these problems, numerous *policy miners* have been proposed [12, 18, 31, 33, 43, 52, 56, 77]. We describe some of them in Section 3.1. Policy miners are algorithms that receive as input the current *permission assignment*, which is a relation between the set of users and the set of permissions. The permission assignment might be given as an access control matrix or a log of access requests showing the access decisions previously made for each request. It describes the organization's implemented knowledge on which permissions should be assigned to which users. The miner then constructs a policy that is as consistent as possible with the permission assignment and can be expressed using the organization's policy language.

A policy miner aims to solve the two problems mentioned above. First, it can mine succinct policies that grant permissions consistent with the given permission assignment. Second, policy miners can mine policies that assign only those permissions that users necessarily need. An administrator can then compare the mined policy with the currently implemented policy in order to detect permissions that are granted by the current policy, but that are not being exercised by the users. Policy administrators can then inspect those permissions and decide if they are necessary for those users.

The problem of policy mining is defined as follows. Given a permission assignment and an objective function, compute a policy that minimizes the objective function. Usually, objective functions measure how well a policy fits a permission assignment and how complex a policy is. We give examples of objective functions later in Sections 5 and 8.

2.2 Quality criteria for policy miners

Policy miners can be regarded as machine-learning algorithms. Therefore, they are evaluated by the quality of the policies they mine, and here two criteria are used:

Generalization [18, 30, 56]. A mined policy should not only authorize requests consistent with the given permission assignment. It must also correctly decide what *other* permissions should be granted to users who perform similar functions in the organization. This is particularly important when mining from logs. For example, if most of students in a university have requested and been granted access to a computer room, then the mined policy should grant all students access to the computer room rather than just to those who previously requested access to it. For a formal definition of generalization, we refer to previous work and standard references in machine learning [9, 18, 30]. One popular machine-learning method to evaluate generalization is cross-validation [9, 32].

Complexity [12, 77]. A mined policy should not be unnecessarily complex, as the policies are usually reviewed and audited by humans. This is especially important when mining with the goal of refactoring an existing policy or migrating to a new policy language. However, there is no standard formalization of a policy's complexity, not even for established policy languages like RBAC or ABAC. Each previous work has defined its own metrics to quantify complexity [18, 29, 76, 77]. We discuss some of these metrics in Section 8 and show how Unicorn is able to work with all of them.

3 THE PROBLEM OF DESIGNING POLICY MINERS

3.1 Status quo: specialized solutions

Numerous policy languages exist for specifying access control policies, which fulfill different organizational requirements. Moreover, new languages are continually being proposed. Some of them formulate new concepts, like extensions of RBAC that can express temporal and spatial constraints [1, 13, 15, 20, 49, 60, 69]. Other languages facilitate policy specification in specialized settings such as distributed systems [34, 70] or social networks [28].

Motivated by the practical problem of maintaining access control configurations, researchers have proposed policy miners for a variety of policy languages. Moreover, for some policy languages, these miners optimize different objectives. For example, initial RBAC miners mined policies with a minimal number of roles [51, 63, 72, 74, 80]. Subsequent miners mined policies that are as consistent as possible with the user-attribute information [30, 56, 75].

The development of policy miners is non-trivial and generally requires sophisticated combinatorial and machine-learning algorithms. Recent ABAC miners have used association rule mining [18] and classification trees [14]. The most effective RBAC miners use deterministic annealing [30] and latent Dirichlet allocation [56].

The proposed miners are so specialized that it is usually unclear how to apply them to other policy languages or even to extensions of the languages for which they were conceived. For example, different extensions of RBAC that support spatio-temporal constraints have been proposed over the last two decades, e.g., [1, 13, 15, 20, 49, 60, 69]. However, not a single miner has been proposed for these extensions. Miners have only recently emerged that mine RBAC policies with constraints, albeit only temporal ones [53, 54, 66]. As a result, if an organization wants to use a specialized policy language, it must invent its own policy miner, which is challenging and time-consuming.

3.2 Alternative: A universal method

To facilitate the development of policy miners, we propose a new method, *universal access control policy mining* (Unicorn). With this method, organizations no longer need to spend substantial effort designing specialized policy miners for their unique and specific policy languages; they only need to perform the following tasks (see also Figure 1). First, they specify a *template formula* φ for the organization's policy language. We explain later in Section 4 what a template formula is. Second, they specify an objective function.

Finally, they implement the miner as indicated by the algorithm template in Section 7. We formalize these tasks in the next sections.

4 A UNIVERSAL POLICY LANGUAGE

In order to obtain a universal method, we need a framework for specifying policy languages. We choose *many-sorted first-order logic* [23, 24], which has been used to model and reason about numerous policy languages, e.g. [3, 19, 40, 71].

Let Γ be a policy language for which we want to design a policy miner. In this section we explain the first task: the miner designer must specify a template formula φ_Γ for Γ. This is a first-order formula that fulfills some conditions that we explain later in Definition 5. We show how Γ can be identified with $\varphi_\Gamma \in \mathcal{L}$ and how policies in Γ can be identified with *interpretation functions* that interpret φ_Γ's symbols. We thereby reduce the problem of designing a policy miner to designing an algorithm that searches for a particular interpretation function.

We start by recalling first-order logic (Section 4.1). Then we provide some intuition on template formulas using RBAC (Section 4.2). Afterwards, we propose a fragment $\mathcal{L}$ of first-order logic that is powerful enough to contain template formulas for a variety of policy languages like RBAC, ABAC, and an expressive fragment of XACML (Section 4.3). We then define template formulas (Section 4.4) and give an example of a template formula for RBAC (Section 4.5).

4.1 Background in first-order logic

We provide here an overview of basic many-sorted first-order logic and conventions we employ. The reader familiar with logic can read this section lightly. We work only with *finite* first-order structures. That is, structures whose carrier sets are finite. Later, in our examples, we will see that finite structures are still powerful enough to model practical scenarios, as organizations do not need to handle infinite sets. Even for the case of strings and integers, organizations often only use a finite subset of them.

Definition 1. A *signature* is a tuple $(\mathbb{S}, \mathbb{R}, \mathbb{F}, \mathbb{V})$ fulfilling the following, where $\mathbb{S}$ is a finite non-empty set of *sorts*, $\mathbb{R}$ is a finite non-empty set of *relation symbols*, $\mathbb{F}$ is a finite non-empty set of *function symbols*, and $\mathbb{V}$ is a countable set of *variables*.

Each relation and each function symbol has an associated *type*, which is a sequence of sorts. Furthermore, we assume the existence of two sorts **USERS**, **PERMS** $\in \mathbb{S}$, denoting the users and the permissions in the organization, respectively. We also assume the existence of the sorts **BOOL**, **INTS**, **STRS**, which represent Boolean values, integers, and strings, respectively. □

We denote sorts with **CAPITAL BOLD** letters, relation symbols with *CAPITAL ITALIC* letters, and function symbols and variables with *small italic* letters. To agree with standard notation, we write a relation symbol's type $(\mathbf{S}_1, \dots, \mathbf{S}_k)$ as $\mathbf{S}_1 \times \dots \times \mathbf{S}_k$ instead. We write a function's symbol's type $(\mathbf{S}_1, \dots, \mathbf{S}_k)$ as $\mathbf{S}_1 \times \dots \times \mathbf{S}_{k-1} \to \mathbf{S}_k$ instead. We allow $k = 1$ and, in that case, we call function symbols *constant symbols*. We denote constant symbols with small serif letters.

Definition 2. Let Σ be a signature. We define *(first-order) terms* as those expressions built from Σ's variables and function symbols in the standard way. We also define *(first-order) formulas* as those expressions obtained from terms by using relation symbols, terms, and logical operators in the standard way. □

We only allow well-typed terms and formulas and associate to every term a type in the standard way. In addition, we consider only quantifier-free formulas. For a formula φ, if $\{x_1, \dots, x_n\}$ is the set of all variables occurring in it, then we sometimes write $\varphi(x_1, \dots, x_n)$ instead of φ to clarify which variables occur in φ.

Definition 3. Let Σ be a signature. A Σ*-structure* is a pair $\mathbb{K} = (\mathfrak{S}, \mathfrak{I})$. Here, $\mathfrak{S}$ is a function mapping each sort $\mathbf{S}$ in Σ to a finite non-empty set $\mathbf{S}^{\mathfrak{S}}$, called ***S**'s carrier set.* $\mathfrak{S}$ must map **BOOL**, **INTS**, and **STRS** to the sets of Boolean values, a finite set of integers, and a finite set of strings, respectively. $\mathfrak{I}$ is a function mapping (i) each relation symbol R in Σ of type $\mathbf{S}_1 \times \dots \times \mathbf{S}_k$ to a relation $R^{\mathfrak{I}} \subseteq \mathbf{S}_1^{\mathfrak{S}} \times \dots \times \mathbf{S}_k^{\mathfrak{S}}$ and (ii) each function symbol f in Σ of type $\mathbf{S}_1 \times \dots \times \mathbf{S}_{k-1} \to \mathbf{S}_k$ to a function $f^{\mathfrak{I}} : \mathbf{S}_1^{\mathfrak{S}} \times \dots \times \mathbf{S}_{k-1}^{\mathfrak{S}} \to \mathbf{S}_k^{\mathfrak{S}}$. In particular, a constant symbol of sort $\mathbf{S}$ is mapped to an element in $\mathbf{S}^{\mathfrak{S}}$. For any symbol W in Σ, we call $W^{\mathfrak{I}}$, *$\mathbb{K}$'s interpretation of W.* The function $\mathfrak{I}$ is called an *interpretation function.* □

When Σ is irrelevant or clear from the context, we simply say structure instead of Σ-structure. We denote elements of carrier sets with small serif letters like a and b.

Let $(\mathfrak{S}, \mathfrak{I})$. The interpretation function $\mathfrak{I}$ gives rise in the standard way to a function that maps any formula $\varphi(x_1, \dots, x_n)$, with x_i of sort $\mathbf{W}_i$, to a relation $\varphi^{\mathfrak{I}} \subseteq \mathbf{W}_1^{\mathfrak{S}} \times \dots \times \mathbf{W}_n^{\mathfrak{S}}$. For $(\mathrm{a}_1, \dots \mathrm{a}_n) \in \mathbf{W}_1^{\mathfrak{S}} \times \dots \times \mathbf{W}_n^{\mathfrak{S}}$, $\varphi^{\mathfrak{I}}(\mathrm{a}_1, \dots \mathrm{a}_n)$ holds if the formula φ evaluates to true after replacing each x_i with a_i.

4.2 Motivating example

We present an example of a template formula φ_N^{RBAC} for the language Γ_N of all RBAC policies with at most N roles. We then show that *every RBAC policy in* Γ *can be identified with an interpretation function.* With this example, we provide some intuition on an argument we give later in Section 4.4: mining a policy in a policy language Γ is equivalent to searching for an interpretation function that interprets the symbols occurring in a template formula for Γ.

Definition 4. An *RBAC policy* is a tuple $\pi = (U, Ro, P, Ua, Pa)$. U and P are non-empty sets denoting, respectively, the sets of users and permissions in an organization. Ro is a set denoting the organization's roles. $Ua \subseteq U \times Ro$ and $Pa \subseteq Ro \times P$ are binary relations. The policy π *assigns* a permission $\mathrm{p} \in P$ to a user $\mathrm{u} \in U$ if $(\mathrm{u}, \mathrm{p}) \in Ua \circ Pa$ (i.e., if there is a role $\mathrm{r} \in Ro$ such that $(\mathrm{u}, \mathrm{r}) \in Ua$ and $(\mathrm{r}, \mathrm{p}) \in Pa$). □

Consider the language Γ_N of RBAC policies with at most N roles. We now present a template formula for Γ_N. We only provide some intuition here and give formal justifications in Section 4.5. Let Σ be a signature with two relation symbols UA and PA of types **USERS** $\times$ **ROLES** and **ROLES** $\times$ **PERMS**, respectively. Let

$$\varphi_N^{RBAC}(u, p) := \bigvee_{i \le N} \left(UA(u, \mathrm{r}_i) \wedge PA(\mathrm{r}_i, p) \right). \quad (1)$$

Here, u and p are variables of sorts **USERS** and **PERMS**, respectively, and r_i, for $i \le N$, is a constant of sort **ROLES**. We now make two observations about $\varphi_N^{RBAC}(u, p)$.

1) Each RBAC policy in Γ_N corresponds to at least one interpretation function. Note that for any Σ-structure $\mathbb{K} = (\mathfrak{S}, \mathfrak{I})$, the tuple

$$\pi_{\mathbb{K}} = \left(\mathbf{USERS}^{\mathfrak{S}}, \mathbf{ROLES}^{\mathfrak{S}}, \mathbf{PERMS}^{\mathfrak{S}}, UA^{\mathfrak{I}}, PA^{\mathfrak{I}}\right) \quad (2)$$

is an RBAC policy. Conversely, one can show that every RBAC policy in Γ_N can be associated with a Σ-structure. Observe now that, when an organization wants to mine an RBAC policy, $\mathfrak{S}$ is already known. Indeed, the organization knows the set of users and permissions. It may not known the set of roles, but it can deduce them from $UA^{\mathfrak{I}}$ and $PA^{\mathfrak{I}}$, once it knows $\mathfrak{I}$. Analogously, for all policy languages we studied, we observed that $\mathfrak{S}$ was always known by the organization. Therefore, we always assume $\mathfrak{S}$ given and fixed and we conclude that every RBAC policy in Γ_N corresponds to at least one interpretation function.

2) The formula φ_N^{RBAC} describes Γ_N's semantics. More precisely, if $\pi_{\mathbb{K}}$ has at most N roles, then for any user u in $\mathbb{K}$ and any p in $\mathbb{K}$: $\pi_{\mathbb{K}}$ assigns p to u iff $\left(\varphi_N^{RBAC}\right)^{\mathfrak{I}}$ (u, p). This follows from two arguments. First, by definition, $\pi_{\mathbb{K}}$ assigns p to u if $(\mathrm{u}, \mathrm{p}) \in UA^{\mathfrak{I}} \circ PA^{\mathfrak{I}}$. Second, $UA^{\mathfrak{I}} \circ PA^{\mathfrak{I}} = \left(\varphi_N^{RBAC}\right)^{\mathfrak{I}}$.

These two observations describe the essence of a template formula. Template formulas define (i) how interpretation functions can represent policies of a policy language and (ii) how a policy (represented by an interpretation function) decides if a permission is assigned to a user.

4.3 Language definition

Template formulas are built from the fragment $\mathcal{L}$ of quantifier-free first-order formulas.

For any signature, we require the organization to specify, for every relation and function symbol, whether it is *rigid* or *flexible*. Rigid symbols are those for which the organization already knows the interpretation function. Flexible symbols are those for which an interpretation function must be found using mining. For example, a function that maps each user to a unique identifier should be modeled with a rigid function symbol, as the organization is not interested in mining new identifiers. In contrast, when mining RBAC policies, one should define a flexible relation symbol to denote the assignment of roles to users, as the organization does not know this assignment and wants to compute it using mining.

Let $\mathbb{K} = (\mathfrak{S}, \mathfrak{I})$ be a structure. We can see $\mathfrak{I}$ as the union of two interpretation functions $\mathfrak{I}_r$ and $\mathfrak{I}_f$, where $\mathfrak{I}_r$ takes as input rigid symbols and $\mathfrak{I}_f$ takes as input flexible symbols. The goal of policy mining is to search for an interpretation function $\mathfrak{I}_f$ for the flexible symbols that minimizes an objective function. It does not need to search for $\mathfrak{S}$ as these function defines the carrier sets for sorts like **USERS** and **PERMS**, which the organization already knows. It does not need to search for $\mathfrak{I}_r$ either. Hence, we assume that $\mathfrak{S}$ and $\mathfrak{I}_r$ are fixed and known to the organization. We also let $U = \mathbf{USERS}^{\mathfrak{S}}$ and $P = \mathbf{PERMS}^{\mathfrak{S}}$. We underline rigid symbols and do not distinguish between $\underline{W}$ and $\underline{W}^{\mathfrak{I}_r}$.

4.4 Template formulas

We now formalize template formulas. Let Γ be a policy language and let $Pol(\Gamma)$ be the set of all policies that can be specified with Γ. Suppose also that the set of access requests is modeled with a set $\mathbf{T}_1^{\mathfrak{S}} \times \ldots \times \mathbf{T}_\ell^{\mathfrak{S}}$, where $\mathbf{T}_1, \ldots, \mathbf{T}_\ell$ are sorts. For example, for RBAC and many other policy languages that we discuss here, the set of requests is $U \times P = \mathbf{USERS}^{\mathfrak{S}} \times \mathbf{PERMS}^{\mathfrak{S}}$.

We assume that the semantics of Γ defines a relation $assign_\Gamma \subseteq Pol(\Gamma) \times \mathbf{T}_1^{\mathfrak{S}} \times \ldots \times \mathbf{T}_\ell^{\mathfrak{S}}$, such that for $(\mathrm{t}_1, \ldots, \mathrm{t}_\ell) \in \mathbf{T}_1^{\mathfrak{S}} \times \ldots \times \mathbf{T}_\ell^{\mathfrak{S}}$ and $\pi \in Pol(\Gamma)$, $(\pi, \mathrm{t}_1, \ldots, \mathrm{t}_\ell) \in assign_\Gamma$ iff π authorizes $(\mathrm{t}_1, \ldots, \mathrm{t}_\ell)$. For example, in RBAC, $(\pi, \mathrm{u}, \mathrm{p}) \in assign_{RBAC}$ iff π assigns p to u.

Definition 5. Let Γ be a policy language and $\varphi(t_1, \ldots, t_\ell)$ be a formula in $\mathcal{L}$, where $t_1, \ldots, t_\ell$ are variables of sorts $\mathbf{T}_1, \ldots, \mathbf{T}_\ell$, respectively. The formula $\varphi(t_1, \ldots, t_\ell)$ is a *template formula for* Γ if there is a function $\mathcal{M}$ such that (i) $\mathcal{M}$ is a surjective function from the set of interpretation functions to $Pol(\Gamma)$ and (ii) for any interpretation function $\mathfrak{I}$ and any request $(\mathrm{t}_1, \ldots, \mathrm{t}_\ell) \in \mathbf{T}_1^{\mathfrak{S}} \times \ldots \times \mathbf{T}_\ell^{\mathfrak{S}}$, we have that $(\mathrm{t}_1, \ldots, \mathrm{t}_\ell) \in \varphi^{\mathfrak{I}}$ iff $(\mathcal{M}(\mathfrak{I}), \mathrm{t}_1, \ldots, \mathrm{t}_\ell) \in assign_\Gamma$. □

The mapping $\mathcal{M}$ provides a correspondence between interpretations and policies. $\mathcal{M}$ guarantees that each policy is represented by at least one interpretation. Therefore, we can search for an interpretation instead of a policy. For this reason, for the rest of the paper, we identify every formula in $\mathcal{L}$ with a *policy language* and also refer to interpretation functions as *policies*.

4.5 Formalizing the example

We now formally define the formula $\varphi_N^{RBAC}(u, p) \in \mathcal{L}$, introduced in Section 4.2, and show that it is a template formula for the language Γ_N of all RBAC policies with at most N roles. Allowing a maximum number of roles is sufficient as one always can estimate a trivial bound on the maximum number of roles in an organization.

Template formula definition: Consider a signature with a sort **ROLES** denoting roles and with two (flexible) binary relation symbols UA and PA of types **USERS** $\times$ **ROLES** and **ROLES** $\times$ **PERMS**, respectively. Define the formula

$$\varphi_N^{RBAC}(u, p) := \bigvee_{i \le N} \left(UA(u, \underline{\mathrm{r}}_i) \wedge PA(\underline{\mathrm{r}}_i, p)\right). \quad (3)$$

Here, $\underline{\mathrm{r}}_i$, for $1 \le i \le N$, is a rigid constant symbol of sort **ROLES** (recall that we underline rigid symbols and denote constant symbols with serif letters). One could also use flexible constant symbols for roles, but, as we see later, the difficulty of implementing the policy miner increases with the number of flexible symbols.

Correctness proof: We now define a mapping $\mathcal{M}$ that proves that φ_N^{RBAC} is a template formula for Γ_N. For any interpretation function $\mathfrak{I}$, let $\mathcal{M}(\mathfrak{I}) = \left(U, \{\underline{\mathrm{r}}_1, \ldots, \underline{\mathrm{r}}_N\}, P, UA^{\mathfrak{I}}, PA^{\mathfrak{I}}\right)$. Observe that $\mathcal{M}(\mathfrak{I})$ is an RBAC policy. Moreover, for $(\mathrm{u}, \mathrm{p}) \in U \times P$, $(\mathrm{u}, \mathrm{p}) \in \left(\varphi_N^{RBAC}\right)^{\mathfrak{I}}$ iff $(\mathrm{u}, \mathrm{p}) \in UA^{\mathfrak{I}} \circ PA^{\mathfrak{I}}$ iff $(\pi_{\mathfrak{I}}, \mathrm{u}, \mathrm{p}) \in assign_{RBAC}$. It is also easy to prove that $\mathcal{M}$ is surjective on the set of all RBAC policies with at most N roles. Hence, we can identify φ_N^{RBAC} with the language of all RBAC policies with at most N roles. □

Example 1. To facilitate understanding $\mathcal{M}$, we show an RBAC policy π and an interpretation function $\mathfrak{I}$ such that $\mathcal{M}(\mathfrak{I}) = \pi$.

Let $N = 2$ and assume that $U = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$ and that $P = \{\mathrm{c}, \mathrm{m}, \mathrm{d}\}$. The permissions in P stand for "create", "modify", and

	r_1	r_2
Alice	×	
Bob	×	
Charlie		×

Table 1: User-assignment relation

	c	m	d
r_1	×	×	
r_2			×

Table 2: Permission-assignment relation

"delete". Let r_1 and r_2 denote two roles. Consider the RBAC policy defined by Tables 1 and 2.

We can define an interpretation function $\mathfrak{I}$ such that $\mathcal{M}(\mathfrak{I})$ corresponds to the RBAC policy above. $\mathfrak{I}$ interprets the relation symbols *UA* and *PA* in the formula $\varphi_N^{RBAC}(u, p)$ as follows. For $u \in U$ and $i \leq 2$, $UA^{\mathfrak{I}}\left(u, \underline{r}_i\right)$ iff (u, r_i) is marked with an × in Table 1. Similarly, for $p \in P$ and $i \leq 2$, $PA^{\mathfrak{I}}\left(\underline{r}_i, p\right)$ iff (r_i, p) is marked with an × in Table 2. □

5 PROBABILITY DISTRIBUTION

Let $\varphi \in \mathcal{L}$ be a policy language. We assume for the rest of the paper that φ has two free variables u and p of sorts **USERS** and **PERMS**, respectively. Our presentation extends in a straightforward way to more general cases.

To design a policy miner using Unicorn one must specify an *objective function L*. This is any function taking two inputs: a permission assignment $Auth \subseteq U \times P$, which is a relation on U and P indicating what permissions each user has, and a policy $\mathfrak{I}$. An objective function outputs a value in $\mathbb{R}^+$ measuring how well $\varphi^{\mathfrak{I}}$ fits *Auth* and other policy requirements. The policy miner designer is in charge of specifying such a function. In Section 8, we give other examples of objective functions.

For illustration, consider the objective function

$$L\left(Auth, \mathfrak{I}; \varphi\right) = \sum_{(u,p)\in U\times P} \left|Auth(u,p) - \varphi^{\mathfrak{I}}(u,p)\right|. \tag{4}$$

Here, we identify the value 1 with the Boolean value `true` and the value 0 with the Boolean value `false`. Observe that $L\left(Auth, \mathfrak{I}; \varphi\right)$ is the size of the symmetric difference of the relations *Auth* and $\varphi^{\mathfrak{I}}$. Hence, lower values for $L\left(Auth, \mathfrak{I}; \varphi\right)$ are better.

The policy miners built with Unicorn are *probabilistic*. They receive as input a permission assignment *Auth* and compute a probability distribution over the set of all policies in a fixed policy language Γ. We use a Bayesian instead of a frequentist interpretation of probability. The probability of a policy $\mathfrak{I}$ does not measure how often $\mathfrak{I}$ is the outcome of an experiment, but rather how strong we believe $\mathfrak{I}$ to be the policy that decided the requests in *Auth*.

We now define, given a permission assignment *Auth*, a probability distribution $\mathbb{P}\left(\cdot \mid Auth\right)$ on policies. We first provide some intuition on $\mathbb{P}\left(\cdot \mid Auth\right)$'s definition and afterwards define it. For a permission assignment *Auth* and a policy $\mathfrak{I}$, we can see $\mathbb{P}\left(\cdot \mid Auth\right)$ as a quantity telling us how much we believe $\mathfrak{I}$ to be the organization's policy, given that *Auth* is the organization's permission assignment.

Policy miners receive as input a permission assignment *Auth* and then search for a policy $\mathfrak{I}^*$ that maximizes $\mathbb{P}\left(\cdot \mid Auth\right)$. Here, $\mathbb{P}\left(\cdot \mid Auth\right)$ is defined as the "most general" distribution that fulfills the following requirement: *for any policy* $\mathfrak{I}$*, the lower* $L\left(Auth, \mathfrak{I}; \varphi\right)$ *is, the more likely* $\mathfrak{I}$ *is.* Following the principle of maximum entropy [45], the most general distribution that achieves this is

$$\mathbb{P}\left(\mathfrak{I} \mid Auth\right) = \frac{\exp\left(-\beta L(Auth, \mathfrak{I}; \varphi)\right)}{\sum_{\mathfrak{I}'} \exp\left(-\beta L(Auth, \mathfrak{I}'; \varphi)\right)}, \tag{5}$$

where $\mathfrak{I}'$ ranges over all policies. Recall that we consider only finite structures. Hence, all our carrier sets are finite, so there are only finitely many policies.

The value $\beta > 0$ is a parameter that the policy miner varies during the search for the most likely policy. The search uses deterministic annealing, an optimization procedure inspired by simulated annealing [46, 61, 62]. In our case, it initially sets β to a very low value, so that all policies are almost equally likely. Then it gradually increases β while, at the same time, searching for the most likely policy. As β increases, those policies that minimize $L(Auth, \cdot\,; \varphi)$ become more likely. In this way, deterministic can escape from low-quality local maxima of $\mathbb{P}\left(\cdot \mid Auth\right)$. When $\beta \to \infty$, only those policies that minimize $L(Auth, \cdot\,; \varphi)$ have a positive probability and the search converges to a local maximum of $\mathbb{P}\left(\cdot \mid Auth\right)$.

We now define the probability distribution given in Equation 5.

Definition 6. For a formula $\varphi \in \mathcal{L}$, we define the probability space $\mathfrak{P}_\varphi = \left(\Omega, 2^\Omega, \mathbb{P}\left(\cdot \mid Auth\right)\right)$ as follows.

- Ω is the set of all interpretation functions (i.e., policies).
- 2^Ω is the set of all subsets of Ω. Since all carrier sets of all sorts are finite (Definition 3), Ω and 2^Ω are finite.
- For $\mathfrak{I} \in \Omega$,

$$\mathbb{P}\left(\mathfrak{I} \mid Auth\right) = \frac{\exp\left(-\beta L(Auth, \mathfrak{I}; \varphi)\right)}{\sum_{\mathfrak{I}'} \exp\left(-\beta L(Auth, \mathfrak{I}'; \varphi)\right)}. \tag{6}$$

Finally, for $O \in 2^\Omega$, let $\mathbb{P}\left(O \mid Auth\right) = \sum_{\mathfrak{I}\in O} \mathbb{P}\left(\mathfrak{I} \mid Auth\right)$.

□

The following theorem proves that $\mathbb{P}\left(\cdot \mid Auth\right)$ is the "most general" distribution that fulfills the requirement mentioned above. More precisely, $\mathbb{P}\left(\cdot \mid Auth\right)$ is the maximum-entropy probability distribution where the probability of a policy $\mathfrak{I}$ increases whenever $L(Auth, \mathfrak{I}; \varphi)$ decreases [39, 68].

Theorem 1. *$\mathbb{P}\left(\cdot \mid Auth\right)$ is the distribution P on policies that maximizes P's entropy and is subject to the following constraints.*

- *$\sum_{\mathfrak{I}} P\left(\mathfrak{I}\right) L(Auth, \mathfrak{I}; \varphi) \leq \ell$, for some fixed bound ℓ.*
- *If $\beta > 0$, then $P\left(\mathfrak{I}\right) > P\left(\mathfrak{I}'\right)$, for any two policies $\mathfrak{I}$ and $\mathfrak{I}'$ with $L(Auth, \mathfrak{I}; \varphi) < L(Auth, \mathfrak{I}'; \varphi)$.*

Proof. It suffices to drop the second constraint and use Lagrange multipliers to verify that $\mathbb{P}\left(\mathfrak{I} \mid Auth\right)$ is the optimal distribution. Observe that $\mathbb{P}\left(\mathfrak{I} \mid Auth\right)$ satisfies the second constraint. □

Example 2. We illustrate the probability distribution defined above for the language of all RBAC policies with at most N roles, defined in Section 4.5. For simplicity, we fix $N = 2$ in this example. Assume that $U = \{\text{Alice}, \text{Bob}, \text{Charlie}\}$ and that $P = \{c, m, d\}$, as defined in Example 1. Assume given a permission assignment *Auth* and two policies $\mathfrak{I}_1$ and $\mathfrak{I}_2$ as shown in Tables 3–9.

Recall that $\left(\varphi_N^{RBAC}\right)^{\mathfrak{I}_1}$ and $\left(\varphi_N^{RBAC}\right)^{\mathfrak{I}_2}$ are the permission assignments induced by $\mathfrak{I}_1$ and $\mathfrak{I}_2$, respectively. Observe that $\left(\varphi_N^{RBAC}\right)^{\mathfrak{I}_1}$ and *Auth* differ by one entry, whereas $\left(\varphi_N^{RBAC}\right)^{\mathfrak{I}_2}$ and *Auth* differ by two. Hence, $L\left(Auth, \mathfrak{I}_1; \varphi_N^{RBAC}\right) = 1 < 2 = L\left(Auth, \mathfrak{I}_2; \varphi_N^{RBAC}\right)$. As a result, for any $\beta > 0$, we get that $\mathbb{P}(\mathfrak{I}_1 \mid Auth) = \frac{\exp(-\beta)}{Z} > \frac{\exp(-2\beta)}{Z} = \mathbb{P}(\mathfrak{I}_2 \mid Auth)$, where $Z = \sum_{\mathfrak{I}'} \exp(-\beta L(Auth, \mathfrak{I}'; \varphi))$. □

	c	m	d
Alice	×	×	
Bob	×	×	
Charlie	×		×

Table 3: *Auth*

	$\underline{r}_1^{\mathfrak{I}_1}$	$\underline{r}_2^{\mathfrak{I}_1}$
Alice	×	
Bob	×	
Charlie		×

Table 4: $UA^{\mathfrak{I}_1}$

	$\underline{r}_1^{\mathfrak{I}_2}$	$\underline{r}_2^{\mathfrak{I}_2}$
Alice	×	
Bob	×	
Charlie		×

Table 5: $UA^{\mathfrak{I}_2}$

	c	m	d
$\underline{r}_1^{\mathfrak{I}_1}$	×	×	
$\underline{r}_2^{\mathfrak{I}_1}$			×

Table 6: $PA^{\mathfrak{I}_1}$

	c	m	d
$\underline{r}_1^{\mathfrak{I}_2}$		×	
$\underline{r}_2^{\mathfrak{I}_2}$	×		×

Table 7: $PA^{\mathfrak{I}_2}$

	c	m	d
Alice	×	×	
Bob	×	×	
Charlie			×

Table 8: $\left(\varphi_N^{RBAC}\right)^{\mathfrak{I}_1}$

	c	m	d
Alice		×	
Bob		×	
Charlie	×		×

Table 9: $\left(\varphi_N^{RBAC}\right)^{\mathfrak{I}_2}$

6 APPLYING MEAN-FIELD APPROXIMATION

The policy miner that is built with Unicorn is an algorithm that receives as input a permission assignment *Auth* and computes a policy $\mathfrak{I}$ that approximately maximizes $\mathbb{P}(\cdot \mid Auth)$, while letting $\beta \to \infty$. Since computing $\mathbb{P}(\cdot \mid Auth)$ is intractable, we use *mean-field approximation* [9], a technique that defines an iterative procedure to approximate $\mathbb{P}(\cdot \mid Auth)$ with a distribution $q(\cdot)$. It turns out that computing and maximizing $q(\cdot)$ is much easier than computing and maximizing $\mathbb{P}(\cdot \mid Auth)$. The policy miner is then an algorithm implementing the computation of q and its maximization.

We next introduce some random variables that help to measure the probability that a policy authorizes a particular request $(\mathsf{u}, \mathsf{p}) \in U \times P$ (Section 6.1). Afterwards, we present the approximating distribution q (Section 6.2).

6.1 Random variables

Recall that the sample space Ω of the distribution $\mathbb{P}(\cdot \mid Auth)$ from Definition 6 is the set of all policies $\mathfrak{I}$. Let $\mathfrak{X}$ be a random variable mapping $\mathfrak{I} \in \Omega$ to $\mathfrak{I}$. Although $\mathfrak{X}$'s definition is trivial, it will help us to understand other random variables that we introduce later. We can understand $\mathfrak{X}$ as an "unknown policy" and, for a policy $\mathfrak{I}$, the probability statement $\mathbb{P}(\mathfrak{X} = \mathfrak{I} \mid Auth)$ measures how much we believe that $\mathfrak{X}$ is actually $\mathfrak{I}$, given that the organization's permission assignment is *Auth*. By definition, $\mathbb{P}(\mathfrak{X} = \mathfrak{I} \mid Auth) = \mathbb{P}(\mathfrak{I} \mid Auth)$.

Definition 7. Let $\varphi \in \mathcal{L}$ and let W be a flexible relation symbol occurring in φ of type $\mathbf{S}_1 \times \ldots \times \mathbf{S}_k$ and let f be a flexible function symbol occurring in φ of type $\mathbf{S}_1 \times \ldots \times \mathbf{S}_k \to \mathbf{S}$. Let $(\mathsf{a}_1, \ldots, \mathsf{a}_k) \in \mathbf{S}_1^{\mathfrak{S}} \times \ldots \times \mathbf{S}_k^{\mathfrak{S}}$. Recall that $\mathfrak{S}$ maps sorts to carrier sets. We define the random variable $W^{\mathfrak{X}}(\mathsf{a}_1, \ldots, \mathsf{a}_k) : \Omega \to \{0, 1\}$ that maps $(Auth, \mathfrak{I}) \in \Omega$ to $W^{\mathfrak{I}}(\mathsf{a}_1, \ldots, \mathsf{a}_k) \in \{0, 1\}$. Similarly, we define the random variable $f^{\mathfrak{X}}(\mathsf{a}_1, \ldots, \mathsf{a}_k) : \Omega \to \mathbf{S}^{\mathfrak{S}}$ that maps $(Auth, \mathfrak{I}) \in \Omega$ to $f^{\mathfrak{I}}(\mathsf{a}_1, \ldots, \mathsf{a}_k) \in \mathbf{S}^{\mathfrak{S}}$. We call these random variables *random facts of* φ. □

Example 3. Let us examine some random facts of the formula φ_N^{RBAC} from Example 2. One such random fact is $UA^{\mathfrak{X}}\left(\mathsf{Alice}, \underline{r}_1\right)$, which can take the values 0 and 1, so $UA^{\mathfrak{X}}\left(\mathsf{Alice}, \underline{r}_1\right)$ is a Bernoulli random variable whose probability distribution is defined by

$$\mathbb{P}\left(UA^{\mathfrak{X}}\left(\mathsf{Alice}, \underline{r}_1\right) = 1 \mid Auth\right) = \mathbb{P}\left(\left\{\mathfrak{I} \in \Omega \mid UA^{\mathfrak{I}}\left(\mathsf{Alice}, \underline{r}_1\right) = 1\right\} \mid Auth\right). \quad (7)$$

More generally, the set of random facts for φ_N^{RBAC} is

$$\mathfrak{F}\left(\varphi_N^{RBAC}\right) = \left\{UA^{\mathfrak{X}}\left(\mathsf{u}, \underline{r}_i\right) \mid \mathsf{u} \in U, i \leq N\right\} \cup \left\{PA^{\mathfrak{X}}\left(\underline{r}_i, \mathsf{p}\right) \mid \mathsf{p} \in P, i \leq N\right\}. \quad (8)$$

If we set $N = 2$ and replace each random fact with a Boolean value, as indicated by Tables 4 and 5, then we get an RBAC policy.

Just like a statement of the form $\mathbb{P}(\mathfrak{X} = \mathfrak{I} \mid Auth)$ quantifies how much we believe that $\mathfrak{X} = \mathfrak{I}$ for a given *Auth*, a statement of the form $\mathbb{P}\left(UA^{\mathfrak{X}}\left(\mathsf{Alice}, \underline{r}_1\right) = 1 \mid Auth\right)$ quantifies how much we believe that role $\underline{r}_1$ is assigned to Alice for a given *Auth*. □

Observation 1. Since we assume carrier sets to be finite, a random fact always has a discrete distribution. In particular, random facts built from flexible relation symbols have Bernoulli distributions as they can only take Boolean values. □

We usually denote random facts with Fraktur letters $\mathfrak{f}, \mathfrak{g}, \ldots$ For a random fact $\mathfrak{f}$ of the form $W^{\mathfrak{X}}(\mathsf{a}_1, \ldots, \mathsf{a}_k)$, we denote by $\mathfrak{f}^{\mathfrak{I}}$ the Boolean value $W^{\mathfrak{I}}(\mathsf{a}_1, \ldots, \mathsf{a}_k)$. Similarly, when $\mathfrak{f}$ is of the form $f^{\mathfrak{X}}(\mathsf{a}_1, \ldots, \mathsf{a}_k)$, we denote by $\mathfrak{f}^{\mathfrak{I}}$ the value $f^{\mathfrak{I}}(\mathsf{a}_1, \ldots, \mathsf{a}_k)$. Finally, we denote $\mathfrak{f}$'s range with $Range(\mathfrak{f})$.

For a policy language $\varphi \in \mathcal{L}$, we denote by $\mathfrak{F}(\varphi)$ the set of all random facts of φ. Recall that we assume all our carrier sets to be finite, so $\mathfrak{F}(\varphi)$ is finite.

Observe that, for any formula $\varphi \in \mathcal{L}$, replacing each random fact $\mathfrak{f}$ in $\mathfrak{F}(\varphi)$ with a value in $Range(\mathfrak{f})$ yields a policy. Hence, a policy miner, instead of searching for a policy $\mathfrak{I}$, it just searches for adequate values for all random facts in $\mathfrak{F}(\varphi)$. We formalize this in Lemma 1, whose proof is in the full version.

Lemma 1. For a policy language $\varphi \in \mathcal{L}$,

$$\mathbb{P}(\mathfrak{I} \mid Auth) = \mathbb{P}\left(\left(\mathfrak{f}^{\mathfrak{X}}\right)_{\mathfrak{f}\in\mathfrak{F}(\varphi)} = \left(\mathfrak{f}^{\mathfrak{I}}\right)_{\mathfrak{f}\in\mathfrak{F}(\varphi)} \mid Auth\right). \quad (9)$$

We denote by $h(\cdot)$ the function $\mathbb{P}\left(\left(\mathfrak{f}^{\mathfrak{X}}\right)_{\mathfrak{f}\in\mathfrak{F}(\varphi)} = \cdot \mid Auth\right)$. To avoid cluttered notation, we write $h(\mathfrak{I})$ instead of $h\left(\left(\mathfrak{f}^{\mathfrak{I}}\right)_{\mathfrak{f}\in\mathfrak{F}(\varphi)}\right)$.

We conclude this section by defining some other useful random variables. Recall that $\mathfrak{X}$ is the random variable that maps $\mathfrak{I} \in \Omega$ to $\mathfrak{I}$.

Definition 8. For $(\mathrm{u}, \mathrm{p}) \in U \times P$, $\varphi \in \mathcal{L}$, we define the random variable $\varphi^{\mathfrak{X}}(\mathrm{u}, \mathrm{p}) : \Omega \to \{0, 1\}$ as the function mapping $(Auth, \mathfrak{I})$ to $\varphi^{\mathfrak{I}}(\mathrm{u}, \mathrm{p})$. □

Definition 9. For $\varphi \in \mathcal{L}$, $Auth \subseteq U \times P$, we define the following random variable:

$$L(Auth, \mathfrak{X}; \varphi) := \sum_{(\mathrm{u},\mathrm{p})\in U\times P} \left| Auth(\mathrm{u}, \mathrm{p}) - \varphi^{\mathfrak{X}}(\mathrm{u}, \mathrm{p}) \right|. \quad (10)$$

□

6.2 Approximating the distribution

A mean-field approximation of the probability distribution h is a distribution q defined by

$$q(\mathfrak{I}) := \prod_{\mathfrak{f}\in\mathfrak{F}(\varphi)} q_{\mathfrak{f}}\left(\mathfrak{f}^{\mathfrak{I}}\right), \quad (11)$$

where $q_{\mathfrak{f}} : Range(\mathfrak{f}) \to [0, 1]$ is a probability mass function for $\mathfrak{f}$. Hence, $\sum_{\mathrm{b}\in Range(\mathfrak{f})} q_{\mathfrak{f}}(\mathrm{b}) = 1$. For $\mathrm{b} \in Range(\mathfrak{f})$, the value $q_{\mathfrak{f}}(\mathrm{b})$ denotes the probability, according to $q_{\mathfrak{f}}$, that $\mathfrak{f} = \mathrm{b}$.

Observe that $q(\mathfrak{I})$'s factorization implies that the set of random facts is mutually independent. This is not true in general, as h may not be necessarily factorized like q. This independence assumption is imposed by mean-field theory to facilitate computations. Our experimental results in Section 10 show that, despite this approximation, we still mine high quality policies.

According to mean-field theory, the distributions $\left\{\widehat{q}_{\mathfrak{f}} \mid \mathfrak{f} \in \mathfrak{F}(\varphi)\right\}$ that make q best approximate h are given by

$$\widehat{q}_{\mathfrak{f}}(\mathrm{b}) = \frac{\exp\left(-\beta\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[L(Auth, \mathfrak{X}; \varphi)]\right)}{\sum_{\mathrm{b}'\in Range(\mathfrak{f})} \exp\left(-\beta\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}'}[L(Auth, \mathfrak{X}; \varphi)]\right)}, \quad (12)$$

where $\mathrm{b} \in Range(\mathfrak{f})$ and $\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[L(Auth, \mathfrak{X}; \varphi)]$ is the expectation of $L(Auth, \mathfrak{X}; \varphi)$ after replacing every occurrence of the random fact $\mathfrak{f}$ with b [9]. This expectation is computed using the distribution q. Therefore,

$$\begin{aligned}&\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[L(Auth, \mathfrak{X}; \varphi)] \\ &\quad = \sum_{\mathfrak{I}} \prod_{\substack{\mathfrak{g}\in\mathfrak{F}(\varphi)\\ \mathfrak{g}\neq\mathfrak{f}}} \widehat{q}_{\mathfrak{g}}\left(\mathfrak{g}^{\mathfrak{I}}\right) \left(L(Auth, \mathfrak{I}; \varphi)\{\mathfrak{f} \mapsto \mathrm{b}\}\right). \quad (13)\end{aligned}$$

Here, $L(Auth, \mathfrak{I}; \varphi)\{\mathfrak{f} \mapsto \mathrm{b}\}$ is obtained from $L(Auth, \mathfrak{I}; \varphi)$ by replacing $\mathfrak{f}$ with b.

Using Lemma 1 and the distribution q, we can approximate $\arg\max_{\mathfrak{I}} \mathbb{P}(\mathfrak{I} \mid Auth)$ by maximizing q.

Observation 2. $\max_{\mathfrak{I}} \mathbb{P}(\mathfrak{I} \mid Auth) = \max_{\mathfrak{I}} h(\mathfrak{I}) \approx \max_{\mathfrak{I}} q(\mathfrak{I})$.

The desired miner is then an algorithm that computes q, while letting $\beta \to \infty$, and then computes the policy $\mathfrak{I}^*$ that maximizes q.

7 BUILDING THE POLICY MINER

To compute q, as given by Equation 11, the desired policy miner could use Equation 12 to compute $\widehat{q}_{\mathfrak{f}}$, for each $\mathfrak{f} \in \mathfrak{F}(\varphi)$. Observe, however, that Equation 12 is recursive, since the computation of the expectations on the right hand side requires $\left\{\widehat{q}_{\mathfrak{f}} \mid \mathfrak{f} \in \mathfrak{F}(\varphi)\right\}$, as indicated by Equation 13. This recursive dependency is handled by iteratively computing, for each $\mathfrak{f} \in \mathfrak{F}(\varphi)$, a function $\tilde{q}_{\mathfrak{f}}$ that approximates $\widehat{q}_{\mathfrak{f}}$ [9]. We illustrate this in the step 2a below.

Algorithm 1 gives the pseudocode for computing and maximizing q, which is the essence of the desired policy miner. We give next an overview.

(1) **Initialization (lines 2–3).** Each distribution $\tilde{q}_{\mathfrak{f}}$ is randomly set to an arbitrary function such that $\sum_{\mathrm{b}} \tilde{q}_{\mathfrak{f}}(\mathrm{b}) = 1$.
(2) **Update loop (lines 4–8).** We perform a sequence of iterations that update $\left\{\tilde{q}_{\mathfrak{f}} \mid \mathfrak{f} \in \mathfrak{F}(\varphi)\right\}$ and β. The number T of iterations is fixed before execution.
   (a) **Parameter update (line 5–7).** At each iteration, we compute a random ordering $RS(\mathfrak{F}(\varphi))$ of all the random facts. Then, for each $\mathfrak{f}$ in that order, $\tilde{q}_{\mathfrak{f}}$ is updated to the right-hand side of Equation 12 (lines 6–7), but instead of using $\left\{\widehat{q}_{f} \mid \mathfrak{f} \in \mathfrak{F}(\varphi)\right\}$, we use $\left\{\tilde{q}_{\mathfrak{f}} \mid \mathfrak{f} \in \mathfrak{F}(\varphi)\right\}$ to compute the expectations.
   (b) **Hyper-parameter update (line 8).** After each iteration, we increase β by a factor of α, defined before execution. This approach, originally defined for deterministic annealing, avoids that the algorithm is trapped in a bad local maximum in the early iterations [61, 62].
(3) **Policy computation (line 9).** Finally, we compute the policy $\mathfrak{I}^* = \arg\max_{\mathfrak{I}} q(\mathfrak{I})$. By looking at Equation 11, we see that to maximize q, it suffices to maximize $q_{\mathfrak{f}}$, for every $\mathfrak{f} \in \mathfrak{F}(\varphi)$. Hence, we let $\mathfrak{I}^*$ be the policy that satisfies $\mathfrak{f}^{\mathfrak{I}^*} = \arg\max_{\mathrm{b}\in Range(\mathfrak{f})} \tilde{q}_{\mathfrak{f}}(\mathrm{b})$.

Algorithm 1: The policy miner.

```
1  PolicyMiner($L, Auth, \varphi, \alpha, \beta, T$):
2      for $\mathfrak{f} \in \mathfrak{F}(\varphi)$:
3          Randomly initialize $\tilde{q}_{\mathfrak{f}}$.
4      for $i = 1 \ldots T$:
5          for $\mathfrak{f} \in RS(\mathfrak{F}(\varphi))$:
6              for $b \in Range(\mathfrak{f})$:
7                  $\tilde{q}_{\mathfrak{f}}(\mathrm{b}) \leftarrow \frac{\exp(-\beta\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[L(Auth, \mathfrak{X}; \varphi)])}{\sum_{\mathrm{b}'} \exp(-\beta\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}'}[L(Auth, \mathfrak{X}; \varphi)])}$.
8          $\beta \leftarrow \alpha \times \beta$.
9      Define $\mathfrak{I}^*$ by letting $\mathfrak{f}^{\mathfrak{I}^*} = \arg\max_{\mathrm{b}} \tilde{q}_{\mathfrak{f}}(\mathrm{b})$, for $\mathfrak{f} \in \mathfrak{F}(\varphi)$.
10     return $\mathfrak{I}^*$.
```

Observe that the policy miner requires values for the hyper-parameters α, β, and T as input. Adequate values can be computed using machine-learning methods like grid search [64].

7.1 Simplifying the computation of expectations

One need not be knowledgeable about deterministic annealing or mean-field approximations to implement Algorithm 1 in a standard programming language. The only part requiring knowledge in probability theory is the computation of the expectations in line 7. We now define the notion of *diverse* random variables and show that expectations of some diverse random variables can easily be computed recursively using some basic equalities.

Definition 10. A random variable X is *diverse* if (i) it can be constructed from constant values and random facts using only arithmetic and Boolean operations and (ii) any random fact is used in the construction at most once. □

Example 4. Let $(\mathrm{u},\mathrm{p}) \in U \times P$ and let V, W, and Y be flexible relation symbols. Then $V^{\mathfrak{X}}(\mathrm{u},\mathrm{p}) + W^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$ is diverse, but $V^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\, W^{\mathfrak{X}}(\mathrm{u},\mathrm{p}) + W^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\, Y^{\mathfrak{X}}(\mathrm{u},\mathrm{p}) + V^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\, Y^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$ is not, since each random fact there occurs more than once. □

Corollary 1. Let $\varphi \in \mathcal{L}$ and $(\mathrm{u},\mathrm{p}) \in U \times P$, then $\varphi^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$ is diverse iff every atomic formula that occurs in φ occurs exactly once.

This corollary is a direct consequence of Definition 10. Observe that, for $\varphi \in \mathcal{L}$, one can check in time linear in φ's length that every atomic formula occurring in φ occurs exactly once.

Example 5. Recall the formula φ_N^{RBAC} defined in Section 4.5. Observe that each atomic formula occurs exactly once. Hence, for $(\mathrm{u},\mathrm{p}) \in U \times P$, the random variable $\left(\varphi_N^{RBAC}\right)^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$ is diverse. □

The following lemma, proved in Appendix A, shows how to recursively compute $\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[L(Auth, \mathfrak{X}; \varphi)]$ when $\varphi^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$ is diverse.

Lemma 2. Let $\mathfrak{f}$ and $\mathfrak{g}$ be facts, φ be a formula in $\mathcal{L}$, $(\mathrm{u},\mathrm{p}) \in U \times P$, and $\{\psi_i\}_i \subseteq \mathcal{L}$. Assume that $\varphi^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$ and $\left(\bigwedge_i \psi_i\right)^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$ are diverse. Then the following equalities hold.

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[\mathfrak{g}] = \begin{cases} \mathrm{b} & \text{if } \mathfrak{f} = \mathfrak{g} \text{ and} \\ \sum_{\mathrm{b}\in Range(\mathfrak{g})} \tilde{q}_{\mathfrak{g}}(\mathrm{b})\, \mathrm{b} & \text{otherwise.} \end{cases}$$

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[(\neg\varphi)^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\right] = 1 - \mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\varphi^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\right].$$

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\left(\bigwedge_i \psi_i\right)^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\right] = \prod_i \mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\psi_i^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\right].$$

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[L(Auth, \mathfrak{X}; \varphi)] = \sum_{(\mathrm{u},\mathrm{p})\in U\times P} \left| Auth(\mathrm{u},\mathrm{p}) - \mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\varphi^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\right]\right|.$$

Recall that $\wedge$ and $\neg$ form a complete set of Boolean operators. So one can also use this lemma to compute expectations of diverse random variables of the form $(\varphi \to \psi)^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$ and $(\varphi \vee \psi)^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$.

8 RBAC MINING WITH UNICORN

We explain next how to use Unicorn to build an RBAC miner.

8.1 RBAC policies

We already explained how the formula $\varphi_N^{RBAC} \in \mathcal{L}$ is a template formula for the language of all RBAC policies with at most N roles. To implement Algorithm 1, we only need a procedure to compute $\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[L\left(Auth, \mathfrak{X}; \varphi_N^{RBAC}\right)\right]$. Since, as noted in Example 5, $\left(\varphi_N^{RBAC}\right)^{\mathfrak{X}}(\mathrm{u},\mathrm{p})$ is diverse, we can apply Lemma 2 to show that

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[L\left(Auth, \mathfrak{X}; \varphi_N^{RBAC}\right)\right] = \sum_{(\mathrm{u},\mathrm{p})\in U\times P} \left| Auth(\mathrm{u},\mathrm{p}) - \mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\left(\varphi_N^{RBAC}\right)^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\right]\right|.$$

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\left(\varphi_N^{RBAC}\right)^{\mathfrak{X}}(\mathrm{u},\mathrm{p})\right] = 1 - \prod_{i\le N}\left(1 - \mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[UA^{\mathfrak{X}}(\mathrm{u},\underline{\mathrm{r}}_i)\right]\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[PA^{\mathfrak{X}}(\underline{\mathrm{r}}_i,\mathrm{p})\right]\right),$$

where,

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[UA^{\mathfrak{X}}(\mathrm{u},\underline{\mathrm{r}}_i)\right] = \begin{cases} \mathrm{b} & \text{if } UA^{\mathfrak{X}}(\mathrm{u},\underline{\mathrm{r}}_i) = \mathfrak{f} \\ \sum_{\mathrm{b}} \tilde{q}_{UA^{\mathfrak{X}}(\mathrm{u},\underline{\mathrm{r}}_i)}(\mathrm{b})\, \mathrm{b} & \text{otherwise.} \end{cases}$$

$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[PA^{\mathfrak{X}}(\underline{\mathrm{r}}_i,\mathrm{p})\right]$ is computed analogously.

Observe how the computations of expectations is reduced to a simple rewriting procedure by applying Lemma 2. We can now implement an RBAC miner by implementing Algorithm 1 in a standard programming language and using the results above to compute the needed expectations.

8.2 Simple RBAC policies

The objective function used above has a limitation. When the number of role constants N used by $\varphi_N^{RBAC}(u,p)$ is very large, we might obtain a policy $\tilde{\mathfrak{I}}$ that assigns each role to exactly one user. The role assigned to a user would be assigned all permissions that the user needs. As a result, $L(Auth, \tilde{\mathfrak{I}}; \varphi_N^{RBAC}) = 0$, but $\tilde{\mathfrak{I}}$ is not a desirable policy. We can avoid mining such policies by introducing in the objective function a *regularization term* that measures the complexity of the mined policy $\mathfrak{I}$. A candidate regularization term is:

$$\|\mathfrak{I}\| = \sum_{i\le N}\left(\sum_{\mathrm{u}\in U} UA^{\mathfrak{I}}(\mathrm{u},\underline{\mathrm{r}}_i) + \sum_{\mathrm{p}\in P} PA^{\mathfrak{I}}(\underline{\mathrm{r}}_i,\mathrm{p})\right).$$

Observe that $\|\mathfrak{I}\|$ measures the sizes of the relations $UA^{\mathfrak{I}}$ and $PA^{\mathfrak{I}}$, for $i \le N$, thereby providing a measure of $\mathfrak{I}$'s complexity. We now define the following loss function:

$$L^r_{RBAC}(Auth, \mathfrak{I}) = \lambda\,\|\mathfrak{I}\| + L(Auth, \mathfrak{I}; \varphi).$$

Here $\lambda > 0$ is a trade-off hyper-parameter, which again must be fixed before executing the policy miner and can be estimated using grid search. Note that L^r_{RBAC} penalizes not only policies that substantially disagree with *Auth*, but also policies that are too complex.

The computation of $\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[L^r_{RBAC}(Auth, \mathfrak{X})\right]$ now also requires computing $\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[\|\mathfrak{X}\|]$, where $\|\mathfrak{X}\|$ is the random variable obtained by replacing each occurrence of $\mathfrak{I}$ in $\|\mathfrak{I}\|$ with $\mathfrak{X}$. Fortunately, one can see that $\|\mathfrak{X}\|$ is diverse. Hence, we can use the linearity of expectation and Lemma 2 to compute all needed expectations.

9 MINING SPATIO-TEMPORAL RBAC POLICIES

We now use Unicorn to build the first policy miner for RBAC extensions with spatio-temporal constraints [1, 4, 5, 8, 19, 41]. In policies in these extensions, users are assigned permissions not only according to their roles, but also based on constraints depending on the current time and the user's and the permission's locations. The syntax for specifying these constraints allows for policies like "a user is assigned the role Engineer from Monday through Friday and from 8:00 AM until 5:00 PM" or "the role Engineer is granted permission to access any object within a radius of three miles from the main building."

We present a template formula $\varphi^{st}(t, u, p) \in \mathcal{L}$ for a policy language that we call *spatio-temporal RBAC*. This is an extension of RBAC with a syntax for spatial constraints based on [4, 5] and a syntax for temporal constraints based on temporal RBAC [6].

$$\varphi^{st}(t, u, p) = \bigvee_{i \leq N} \left(\psi_{UA}(t, u, \underline{r}_i) \wedge \psi_{PA}(t, \underline{r}_i, p) \right).$$

Here, we assume the existence of a sort **TIME** and that t is a variable of this sort representing the time when u exercises p. We also assume the existence of a sort **SPACE** that we use to specify spatial constraints. The formulas $\psi_{UA}(t, u, \underline{r}_i)$ and $\psi_{PA}(t, \underline{r}_i, p)$ describe when a user is assigned the role $\underline{r}_i$ and when a permission is assigned to the role $\underline{r}_i$, respectively. We use the rigid constants $\underline{r}_1, \ldots, \underline{r}_N$ to denote roles.

The grammar Γ_{st} below defines the syntax of ψ_{UA} and ψ_{PA}.

$$\begin{aligned} \langle \texttt{cstr\_list} \rangle &::= \langle \texttt{cstr} \rangle \; (\wedge \langle \texttt{cstr} \rangle)* \\ \langle \texttt{cstr} \rangle &::= \langle \texttt{sp\_cstr} \rangle \; (\vee \; \langle \texttt{sp\_cstr} \rangle)* \mid \\ &\quad\;\; \langle \texttt{tmp\_cstr} \rangle \; (\vee \; \langle \texttt{tmp\_cstr} \rangle)* \\ \langle \texttt{sp\_cstr} \rangle &::= (\neg?) \; \underline{isWithin} \left(\underline{Loc}\,(o)\,, \mathsf{d}, \mathsf{b} \right) \\ \langle \texttt{tmp\_cstr} \rangle &::= \psi_{cal}\,(t) \end{aligned}$$

An expression in this grammar is a conjunction of *constraints*, each of which is either a disjunction of *temporal constraints* or a disjunction of *spatial constraints.*

9.1 Modeling spatial constraints

A *spatial constraint* is a (possibly negated) formula of the form $\underline{isWithin}\left(\underline{Loc}\,(o)\,, \mathsf{d}, \mathsf{b}\right)$, where o is a variable of sort **USERS** or **PERMS**, $\underline{Loc}\,(o)$ denotes o's location, d is a flexible constant symbol of a sort whose carrier set is $\mathbb{N}^{\leq M} = \{0, 1, \ldots, \mathsf{M}\}$ (where M is a value fixed in advance), and b is a flexible constant symbol of a sort describing the organization's physical facilities. For example, $\underline{isWithin}\left(\underline{Loc}\,(u)\,, 4, \mathsf{MainBuilding}\right)$ holds when the user represented by u is within 4 space units of the main building.

Intuitively, the formula $\underline{isWithin}\left(\underline{Loc}\,(o)\,, \mathsf{d}, \mathsf{b}\right)$ evaluates whether the entity represented by o is located within d spatial units from b. Observe that a policy miner does not need to compute interpretations for rigid function symbols like $\underline{Loc}$ or rigid relation symbols like $\underline{isWithin}$, since they already have a fixed interpretation.

9.2 Modeling temporal constraints

A *temporal constraint* is a formula $\psi_{cal}\,(t)$ that represents a *periodic expression* [6], which describes a set of time intervals. We give here a simplified overview and refer to the literature for details [6].

Definition 11. A *periodic expression* is a tuple (*yearSet*, *monthSet*, *daySet*, *hourSet*, *hourDuration*) $\in \left(2^{\mathbb{N}}\right)^4 \times \mathbb{N}$. A *time instant* is a tuple $(y, m, d, h) \in \mathbb{N}^4$. The time instant satisfies the periodic expression if $y \in yearSet$, $m \in monthSet$, $d \in daySet$, and there is an $h' \in hourSet$ such that $h' \leq h \leq h' + hourDuration.$ □

Previous works on analyzing temporal RBAC with SMT solvers [40] show that temporal constraints can be expressed as formulas in $\mathcal{L}$. Furthermore, one can verify that any expression in Γ_{st} and, therefore, φ^{st} is in $\mathcal{L}$.

As an objective function, we use $\lambda \left\|\mathfrak{I}\right\| + L\left(Auth, \mathfrak{I}; \varphi^{st}\right)$. Here, $\left\|\mathfrak{I}\right\|$ counts the number of spatial constraints plus the sum of the *weighted structural complexities* of all temporal constraints [66]. For computing expectations, one can show that $\left\|\mathfrak{X}\right\|$ is diverse and that every atomic formula in φ^{st} occurs exactly once. Hence, one can compute all necessary expectations using the linearity of expectation and Lemma 2.

10 EXPERIMENTS

In this section, we experimentally validate two hypotheses. First, using Unicorn, we can build policy miners for a wide variety of policy languages. Second, the policies mined by these miners have as low complexity and high generalization ability as those mined by the state of the art.

10.1 Datasets

Our experiments are divided into the following categories.

Mine RBAC policies from access control matrices. We use three access control matrices from three real organizations, named "healthcare", "firewall", and "americas" [25]. For healthcare, there are 46 users and 46 permissions, for firewall, there are 720 users and 587 permissions, and for americas, there are more than 10,000 users and around 3,500 permissions. We refer to these access control matrices as RBAC1, RBAC2, and RBAC3.

Mine ABAC policies from logs. We use four logs of access requests provided by Amazon for a Kaggle competition in 2013 [43], where participants had to develop mining algorithms that predicted from the logs which permissions must be assigned to which users. We refer to these logs as ABAC1, ABAC2, ABAC3, and ABAC4.

Mine business-meaningful RBAC policies from access control matrices. We use the access control matrix provided by Amazon for the IEEE MLSP 2012 competition [38], available at the UCI machine learning repository [50]. It assigns three types of permissions, named "HOST", "PERM_GROUP", and "SYSTEM_GROUP" to 30,000 users. The number of permissions for each type are approximately 1,700, 6,000, and 20,000, respectively. For each type of permission, we sampled 5,000 users from all 30,000 users and used all permissions of that type to build an access control matrix. We explain in detail how we create these matrices in Appendix D.1. We refer to these matrices as BM-RBAC1, BM-RBAC2, and BM-RBAC3.

Mine XACML policies from access control matrices. We use Continue [27, 47], the most complex set of XACML policies in the literature. We use seven of the largest policies in the set. For each of them, we compute the set of all possible requests and decide which of them are authorized by the policy. We then mine a policy from this set of decided requests. For the simplest policy, there are around 60 requests and for the most complex policy, there are more than 30,000 requests. We call these seven sets of requests XACML1, XACML2, ..., XACML7.

Mine spatio-temporal RBAC policies from logs. There are no publicly available datasets for mining spatio-temporal RBAC policies. Based on policies provided as examples in recent works [4, 5], we created a synthetic policy and a synthetic log by creating 1,000 access requests uniformly at random and evaluating them against the policy. We refer to this log as STARBAC. The synthetic policy is described in Appendix D.2.

10.2 Methodology

For RBAC and ABAC, we mine two policies in the corresponding policy language's syntax. The first one using a miner built using Unicorn and the second one using a state-of-the-art miner. Details on the miners built using Unicorn are given in Sections 8 and 9, and Appendices B and C. For RBAC, we use for comparison the miner presented in [30] and, for ABAC, we use for comparison the miner from [18]. For XACML and spatio-temporal RBAC, there are no other known miners. For business meaningful RBAC, we contacted the authors of miners for this RBAC extension [30, 56], but implementations of their algorithms were not available.

As an objective function we use $\lambda \|\mathfrak{I}\| + L(Auth, \mathfrak{I}; \varphi)$, where λ is a trade-off hyper-parameter, $\|\mathfrak{I}\|$ is the complexity measure defined for $\mathfrak{I}$ in the policy language, and φ is the template formula for the corresponding policy language. The values for the hyper-parameters were computed using grid search.

To evaluate miners for RBAC, BM-RBAC, and XACML, we use 5-fold cross-validation [21, 22, 79]. To measure the mined policy's generalizability, we measure its *true positive rate* (TPR) and its *false positive rate* (FPR) [59]. To measure a mined policy's complexity, we use $\|\mathfrak{I}\|$. To evaluate miners for ABAC and STARBAC, which receive a log instead of an access control matrix as input, we use universal cross-validation [18]. We measure the mined policy's TPR, FPR, precision, and complexity. We considered only those mined policies whose FPR was below 5%.

All policy miners, except the one for BM-RBAC, were developed in Python 3.6 and were executed on machines with 2,8 GHz 8-core CPUs and 32 GB of RAM. The miner for BM-RBAC was developed in Pytorch version 0.4 [58] and executed on an NVIDIA GTX Titan X GPU with 12 GB of RAM. For all policy languages except STARBAC, our experiments finished within 4 hours. For STARBAC, they took 7 hours. We remark that organizations do not need to mine policies on a regular basis, so policies need not be mined in real time [18].

10.3 Results

Figures 2–4 compare, respectively, the TPRs, complexities, and precisions of the policies we mined with those mined by the state of the art across the different datasets with respect to the different policy languages. We make the following observations.

- We mine policies whose TPR is within 5% of the state-of-the-art policies' TPR. For the XACML and STARBAC scenarios, where no other miners exist, we mine policies with a TPR above 75% in all cases.
- In most cases, we mine policies with a complexity lower than the complexity of policies mined by the state of the art.

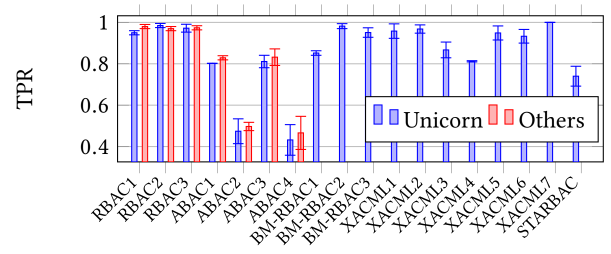

Figure 2: Comparison of the TPRs between policies mined using Unicorn and policies mined by the state of the art across different policy languages. Policies with *higher* TPRs are better at granting permissions to the correct users.

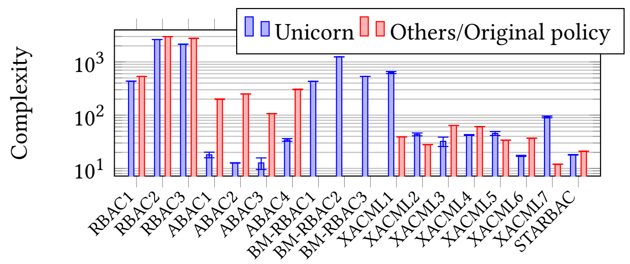

Figure 3: Comparison of the complexities between policies mined using Unicorn and policies mined using the state of the art across different policy languages. Policies with *lower* complexities are better as they are easier to interpret by humans. For XACML and STARBAC, there is no known miner, but we compared the mined policy's complexity with that of the original policy.

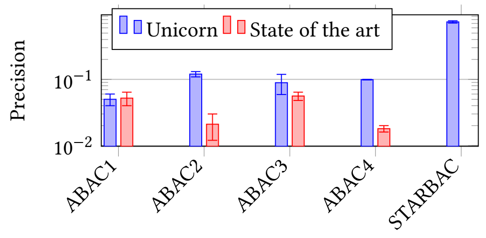

Figure 4: Comparison of the precision between policies mined using Unicorn and policies mined by a state of the art policy miner across different policy languages. Policies with *higher* precision are better as they avoid incorrect authorizations. We only compare the precision of mined policies when mining from logs, as discussed in [18].

- When mining from logs, we mine policies that have a similar or greater precision than those mined by the state of the art, sometimes substantially greater.
- In all cases, we mine policies with an FPR $\leq$ 5% (not shown in the figures).

10.4 Discussion

Our experimental results show that, with the exception of ABAC, all policies we mined attain a TPR of at least 80% in most of the cases. The low TPR in ABAC is due to the fact that the logs contain only 7% of all possible requests [18]. But even in that case, the ABAC miner we built attains a TPR that is within 5% of the TPR attained by the state of the art [18]. Moreover, our ABAC miner mines policies with substantially lower complexity and higher precision. These results support our hypothesis that by using Unicorn we can build competitive policy miners for a wide variety of policy languages.

These results also suggest that the miners built are well-suited for practical use. In this regard, note that policy miners are tools that facilitate the specification and maintenance of policies. They are not intended to replace human policy administrators, especially when the miners work on logs. This is because logs contain just an incomplete view of how permissions should be assigned to users. Very sparse logs, like those used for the experiments on ABAC, contain barely 7% of all possible authorization requests. Hence, we cannot expect policy miners to deduce how all permissions should be assigned from such logs. The policy administrator must review the mined policy and specify how it should decide groups of requests that are not well represented in the log. For this reason, mined policies must also be simple. The main application of policy miners is to reduce the cumbersome effort of manually analyzing logs (or, more generally, permission assignments) and mine policies that *generalize well* (see Section 2.2).

Observe that the mined policies correctly authorize at least 40% of future requests in all cases for ABAC and that in some cases they correctly authorize 80% of all requests. All this with a false positive rate below 5%. This means that the mined policy has already reduced the policy administrator's work by at least 40% and in most of the cases by at least 80%. The administrator now only needs to decide how the policy should decide groups of requests that are not represented in the log.

11 RELATED WORK

11.1 Policy mining

11.1.1 RBAC mining. Early research on policy mining focused on RBAC [25, 48, 73]. The approaches developed used combinatorial algorithms to find, for an assignment of permissions to users, an approximately minimal set of role assignments, e.g., [51, 63, 72, 74, 80]. A major step forward was the use of machine-learning techniques like latent Dirichlet allocation [56] and deterministic annealing [30, 67] to compute models that maximize the likelihood of the given assignment of permission to users. More recent works mine RBAC policies with time constraints [53, 54] and role hierarchies [35, 66], using combinatorial techniques that are specific to the RBAC extension.

Despite the plethora of RBAC miners, there are still many RBAC extensions for which no miner has been developed. A recent survey in role mining [55], covering over a dozen RBAC miners, reports not a single RBAC miner that can mine spatio-temporal constraints, even though there have been several spatio-temporal extensions of RBAC since 2000, e.g., [1, 13, 15, 20, 49, 60, 69], and additional extensions are under way [4, 5]. Unicorn offers a practical solution to mining RBAC policies for these extensions. As illustrated in Section 9, we can now mine spatio-temporal RBAC policies.

11.1.2 Other miners. Miners have recently been proposed for other policy languages like ABAC [18, 76] and ReBAC (Relationship-Based Access Control) [12]. These algorithms use dedicated combinatorial and machine-learning methods to mine policies tailored to the given policy language. Unicorn has the advantage of being applicable to a much broader class of policy languages.

11.2 Interpretable machine learning

Machine-learning algorithms have been proposed that train an *interpretable* model [2, 17, 42, 44, 65] consisting of a set of human-readable rules that describe how an instance is classified. Such algorithms are attractive for policy mining, as policies must not only correctly grant and deny access, they should also be easy to understand.

The main limitation of the rules mined by these models is that they often do not comply with the underlying policy language's syntax. State-of-the-art algorithms in this field [2, 17, 65] produce rules that are simply conjunctions of constraints on the instances' features. This is insufficient for many policy languages, like XACML, where policies can consist of nested subpolicies that are composed with XACML's policy combination algorithms [34].

The main advantage of Unicorn is that it can mine policies that not only correctly grant and deny access in most cases, but are also compliant with a given policy language's syntax, like XACML. Moreover, as illustrated in Section 8.2, one can tailor the objective function so that the policy miner searches for a simple policy.

12 CONCLUSION

The difficulty of specifying and maintaining access control policies has spawned a large and growing number of policy languages with associated policy miners. However, developing such miners is challenging and substantially more difficult than creating a new policy language. This problem is exacerbated by the fact that existing mining algorithms are inflexible in that they cannot be easily modified to mine policies for other policy languages with different features. In this paper, we demonstrated that it is in fact possible to create a universal method for building policy miners that works very well for a wide variety of policy languages.

We validated Unicorn's effectiveness experimentally, including a comparison against state-of-the-art policy miners for different policy languages. In all cases, the miners built using Unicorn are competitive with the state of the art.

As future work, we plan to automate completely the workflow in Figure 1. We envision a *universal policy mining algorithm* based on Algorithm 1 that, given as input the policy language, the permission assignment, and the objective function, automatically computes the probabilistic model and the most likely policy constrained by the given permission assignment.

REFERENCES

[1] Subhendu Aich, Shamik Sural, and Arun K Majumdar. 2007. STARBAC: Spatiotemporal Role Based Access Control. In *OTM Confederated International Conferences" On the Move to Meaningful Internet Systems"*. Springer, 1567–1582.
[2] Elaine Angelino, Nicholas Larus-Stone, Daniel Alabi, Margo Seltzer, and Cynthia Rudin. 2017. Learning Certifiably Optimal Rule Lists. In *Proceedings of the 23rd*

ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. ACM, 35–44.

[3] Konstantine Arkoudas, Ritu Chadha, and Jason Chiang. 2014. Sophisticated Access Control via SMT and Logical Frameworks. *ACM Transactions on Information and System Security (TISSEC)* 16, 4 (2014), 17.

[4] Ameni Ben Fadhel, Domenico Bianculli, and Lionel Briand. 2016. GemRBAC-DSL: a High-level Specification Language for Role-based Access Control Policies. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies.* ACM, 179–190.

[5] Ameni Ben Fadhel, Domenico Bianculli, Lionel Briand, and Benjamin Hourte. 2016. A Model-driven Approach to Representing and Checking RBAC Contextual Policies. In *Proceedings of the Sixth ACM Conference on Data and Application Security and Privacy.* ACM, 243–253.

[6] Elisa Bertino, Piero Andrea Bonatti, and Elena Ferrari. 2001. TRBAC: A temporal Role-based Access Control Model. *ACM Transactions on Information and System Security (TISSEC)* 4, 3 (2001), 191–233.

[7] Smriti Bhatt, Farhan Patwa, and Ravi Sandhu. 2017. ABAC with Group Attributes and Attribute Hierarchies Utilizing the Policy Machine. In *Proceedings of the 2nd ACM Workshop on Attribute-Based Access Control.* ACM, 17–28.

[8] Rafae Bhatti, Arif Ghafoor, Elisa Bertino, and James BD Joshi. 2005. X-GTRBAC: an XML-based Policy Specification Framework and Architecture for Enterprise-wide Access Control. *ACM Transactions on Information and System Security (TISSEC)* 8, 2 (2005), 187–227.

[9] Christopher M Bishop. 2006. *Pattern recognition and machine learning.* springer.

[10] Prosunjit Biswas, Ravi Sandhu, and Ram Krishnan. 2016. Label-based Access Control: an ABAC Model with Enumerated Authorization Policy. In *Proceedings of the 2016 ACM International Workshop on Attribute Based Access Control.* ACM, 1–12.

[11] David M Blei, Alp Kucukelbir, and Jon D McAuliffe. 2017. Variational Inference: A Review for Statisticians. *J. Amer. Statist. Assoc.* 112, 518 (2017), 859–877.

[12] Thang Bui, Scott D Stoller, and Jiajie Li. 2017. Mining Relationship-Based Access Control Policies. *arXiv preprint arXiv:1708.04749* (2017).

[13] Suroop Mohan Chandran and James BD Joshi. 2005. LoT-RBAC: a Location and Time-based RBAC Model. In *International Conference on Web Information Systems Engineering.* Springer, 361–375.

[14] Suresh N Chari and Ian M Molloy. 2016. Generation of Attribute Based Access Control Policy from Existing Authorization System. US Patent 9,264,451.

[15] Liang Chen and Jason Crampton. 2008. On Spatio-temporal Constraints and Inheritance in Role-based Access Control. In *Proceedings of the 2008 ACM Symposium on Information, Computer and Communications Security.* ACM, 205–216.

[16] Yuan Cheng, Khalid Bijon, and Ravi Sandhu. 2016. Extended ReBAC Administrative Models with Cascading Revocation and Provenance Support. In *Proceedings of the 21st ACM on Symposium on Access Control Models and Technologies (SACMAT '16).* ACM, New York, NY, USA, 161–170. https://doi.org/10.1145/2914642.2914655

[17] Peter Clark and Robin Boswell. 1991. Rule Induction with CN2: Some Recent Improvements. In *European Working Session on Learning.* Springer, 151–163.

[18] Carlos Cotrini, Thilo Weghorn, and David Basin. 2018. Mining ABAC Rules from Sparse Logs. In *2018 IEEE European Symposium on Security and Privacy (EuroS&P).* IEEE.

[19] Carlos Cotrini, Thilo Weghorn, David Basin, and Manuel Clavel. 2015. Analyzing First-order Role Based Access Control. In *Computer Security Foundations Symposium (CSF), 2015 IEEE 28th.* IEEE, 3–17.

[20] Xiutao Cui, Yuliang Chen, and Junzhong Gu. 2007. Ex-RBAC: an Extended Role Based Access Control Model for Location-aware Mobile Collaboration System. In *Internet Monitoring and Protection, 2007. ICIMP 2007. Second International Conference on.* IEEE, 36–36.

[21] Massimiliano de Leoni and Wil MP van der Aalst. 2013. Data-aware Process Mining: Discovering Decisions in Processes Using Alignments. In *Proceedings of the 28th annual ACM Symposium on Applied Computing.* ACM, 1454–1461.

[22] AG D'yakonov. 2015. Solution Methods for Classification Problems with Categorical Attributes. *Computational Mathematics and Modeling* 26, 3 (2015), 408–428.

[23] H-D Ebbinghaus, Jörg Flum, and Wolfgang Thomas. 2013. *Mathematical logic.* Springer Science & Business Media.

[24] Herbert Enderton and Herbert B Enderton. 2001. *A mathematical introduction to logic.* Elsevier.

[25] Alina Ene, William Horne, Nikola Milosavljevic, Prasad Rao, Robert Schreiber, and Robert E Tarjan. 2008. Fast exact and heuristic methods for role minimization problems. In *Proceedings of the 13th ACM symposium on Access control models and technologies.* ACM, 1–10.

[26] David F Ferraiolo, Ravi Sandhu, Serban Gavrila, D Richard Kuhn, and Ramaswamy Chandramouli. 2001. Proposed NIST standard for role-based access control. *ACM Transactions on Information and System Security (TISSEC)* 4, 3 (2001), 224–274.

[27] Kathi Fisler, Shriram Krishnamurthi, Leo A Meyerovich, and Michael Carl Tschantz. 2005. Verification and change-impact analysis of access-control policies. In *Proceedings of the 27th International Conference on Software Engineering.* ACM, 196–205.

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

[29] Mario Frank, Joachim M Buhmann, and David Basin. 2010. On the definition of role mining. In *Proceedings of the 15th ACM symposium on Access control models and technologies.* ACM, 35–44.

[30] Mario Frank, Joachim M Buhmann, and David Basin. 2013. Role mining with probabilistic models. *ACM Transactions on Information and System Security (TISSEC)* 15, 4 (2013), 15.

[31] Mario Frank, Andreas P Streich, David Basin, and Joachim M Buhmann. 2009. A probabilistic approach to hybrid role mining. In *Proceedings of the 16th ACM conference on Computer and communications security.* ACM, 101–111.

[32] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. 2001. *The elements of statistical learning.* Vol. 1. Springer series in statistics New York, NY, USA:.

[33] Mayank Gautam, Sadhana Jha, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. 2017. Poster: Constrained Policy Mining in Attribute Based Access Control. In *Proceedings of the 22nd ACM on Symposium on Access Control Models and Technologies.* ACM, 121–123.

[34] Simon Godik and Tim Moses. 2002. Oasis extensible access control markup language (XACML). *OASIS Committee Secification CS-XACML-specification-1.0* (2002).

[35] Qi Guo, Jaideep Vaidya, and Vijayalakshmi Atluri. 2008. The role hierarchy mining problem: Discovery of optimal role hierarchies. In *Computer Security Applications Conference, 2008. ACSAC 2008. Annual.* IEEE, 237–246.

[36] Thomas Hofmann and Joachim M Buhmann. 1997. Pairwise data clustering by deterministic annealing. *IEEE transactions on pattern analysis and machine intelligence* 19, 1 (1997), 1–14.

[37] Vincent C Hu, David Ferraiolo, Rick Kuhn, Arthur R Friedman, Alan J Lang, Margaret M Cogdell, Adam Schnitzer, Kenneth Sandlin, Robert Miller, Karen Scarfone, et al. 2013. Guide to attribute based access control (ABAC) definition and considerations (draft). *NIST special publication* 800, 162 (2013).

[38] IEEE. 2012. 2012 IEEE International workshop on machine learning for signal processing. Amazon data science competition. http://mlsp2012.conwiz.dk/index.php?id=43

[39] Edwin T Jaynes. 1957. Information theory and statistical mechanics. *Physical review* 106, 4 (1957), 620.

[40] Sadhana Jha, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. 2014. Security analysis of temporal RBAC under an administrative model. *Computers & Security* 46 (2014), 154–172.

[41] James BD Joshi. 2004. Access-control language for multidomain environments. *IEEE Internet Computing* 8, 6 (2004), 40–50.

[42] Viktor Jovanoski and Nada Lavrač. 2001. Classification rule learning with APRIORI-C. In *Portuguese Conference on Artificial Intelligence.* Springer, 44–51.

[43] Kaggle. 2013. Amazon.com – Employee access challenge. http://www.kaggle.com/c/amazon-employee-access-challenge

[44] Branko Kavšek and Nada Lavrač. 2006. APRIORI-SD: Adapting association rule learning to subgroup discovery. *Applied Artificial Intelligence* 20, 7 (2006), 543–583.

[45] HK Kesavan and JN Kapur. 1990. Maximum Entropy and Minimum Cross-Entropy Principles: Need for a Broader Perspective. In *Maximum Entropy and Bayesian Methods.* Springer, 419–432.

[46] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. 1983. Optimization by simulated annealing. *science* 220, 4598 (1983), 671–680.

[47] Shriram Krishnamurthi. 2003. The CONTINUE server (or, How I administered PADL 2002 and 2003). In *Practical aspects of declarative languages.* Springer, 2–16.

[48] Martin Kuhlmann, Dalia Shohat, and Gerhard Schimpf. 2003. Role mining-revealing business roles for security administration using data mining technology. In *Proceedings of the eighth ACM symposium on Access control models and technologies.* ACM, 179–186.

[49] Mahendra Kumar and Richard E Newman. 2006. STRBAC–An approach towards spatio-temporal role-based access control.. In *Communication, Network, and Information Security.* 150–155.

[50] M. Lichman. 2013. UCI Machine Learning Repository. Amazon Access Samples Data Set. http://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples

[51] Haibing Lu, Jaideep Vaidya, and Vijayalakshmi Atluri. 2008. Optimal boolean matrix decomposition: Application to role engineering. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on.* IEEE, 297–306.

[52] Barsha Mitra, Shamik Sural, Vijayalakshmi Atluri, and Jaideep Vaidya. 2013. Towards mining of temporal roles. In *IFIP Annual Conference on Data and Applications Security and Privacy.* Springer, 65–80.

[53] Barsha Mitra, Shamik Sural, Vijayalakshmi Atluri, and Jaideep Vaidya. 2015. The generalized temporal role mining problem. *Journal of Computer Security* 23, 1 (2015), 31–58.

[54] Barsha Mitra, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. 2016. Mining temporal roles using many-valued concepts. *Computers & Security* 60 (2016), 79–94.

[55] Barsha Mitra, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. 2016. A survey of role mining. *ACM Computing Surveys (CSUR)* 48, 4 (2016), 50.

[56] Ian Molloy, Youngja Park, and Suresh Chari. 2012. Generative models for access control policies: applications to role mining over logs with attribution. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies.* ACM, 45–56.
[57] Subhojeet Mukherjee, Indrakshi Ray, Indrajit Ray, Hossein Shirazi, Toan Ong, and Michael G. Kahn. 2017. Attribute Based Access Control for Healthcare Resources. In *Proceedings of the 2Nd ACM Workshop on Attribute-Based Access Control (ABAC '17).* ACM, New York, NY, USA, 29–40. https://doi.org/10.1145/3041048.3041055
[58] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. 2017. Automatic differentiation in pytorch. (2017).
[59] David Martin Powers. 2011. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. (2011).
[60] Indrakshi Ray and Manachai Toahchoodee. 2007. A spatio-temporal role-based access control model. In *IFIP Annual Conference on Data and Applications Security and Privacy.* Springer, 211–226.
[61] Kenneth Rose. 1998. Deterministic annealing for clustering, compression, classification, regression, and related optimization problems. *Proc. IEEE* 86, 11 (1998), 2210–2239.
[62] Kenneth Rose, Eitan Gurewitz, and Geoffrey C Fox. 1992. Vector quantization by deterministic annealing. *IEEE Transactions on Information theory* 38, 4 (1992), 1249–1257.
[63] Jürgen Schlegelmilch and Ulrike Steffens. 2005. Role mining with ORCA. In *Proceedings of the tenth ACM symposium on Access control models and technologies.* ACM, 168–176.
[64] Scikit-learn. 2007–2017. Tuning the hyper-parameters of an estimator. http://scikit-learn.org/stable/modules/grid_search.html
[65] Dan Steinberg and Phillip Colla. 2009. CART: classification and regression trees. *The top ten algorithms in data mining* 9 (2009), 179.
[66] Scott D Stoller and Thang Bui. 2016. Mining hierarchical temporal roles with multiple metrics. In *IFIP Annual Conference on Data and Applications Security and Privacy.* Springer, 79–95.
[67] Andreas P Streich, Mario Frank, David Basin, and Joachim M Buhmann. 2009. Multi-assignment clustering for Boolean data. In *Proceedings of the 26th annual international conference on machine learning.* ACM, 969–976.
[68] Y Tikochinsky, NZ Tishby, and Raphael David Levine. 1984. Alternative approach to maximum-entropy inference. *Physical Review A* 30, 5 (1984), 2638.
[69] Manachai Toahchoodee, Indrakshi Ray, Kyriakos Anastasakis, Geri Georg, and Behzad Bordbar. 2009. Ensuring spatio-temporal access control for real-world applications. In *Proceedings of the 14th ACM symposium on Access control models and technologies.* ACM, 13–22.
[70] Petar Tsankov, Srdjan Marinovic, Mohammad Torabi Dashti, and David Basin. 2014. Decentralized composite access control. In *International Conference on Principles of Security and Trust.* Springer, 245–264.
[71] Fatih Turkmen, Jerry den Hartog, Silvio Ranise, and Nicola Zannone. 2015. Analysis of XACML policies with SMT. In *International Conference on Principles of Security and Trust.* Springer, 115–134.
[72] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. 2007. The role mining problem: finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM symposium on Access control models and technologies.* ACM, 175–184.
[73] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. 2010. The role mining problem: A formal perspective. *ACM Transactions on Information and System Security (TISSEC)* 13, 3 (2010), 27.
[74] Jaideep Vaidya, Vijayalakshmi Atluri, and Janice Warner. 2006. RoleMiner: mining roles using subset enumeration. In *Proceedings of the 13th ACM conference on Computer and communications security.* ACM, 144–153.
[75] Zhongyuan Xu and Scott D Stoller. 2012. Algorithms for mining meaningful roles. In *Proceedings of the 17th ACM symposium on Access Control Models and Technologies.* ACM, 57–66.
[76] Zhongyuan Xu and Scott D Stoller. 2014. Mining Attribute-Based Access Control Policies from Logs. In *Data and Applications Security and Privacy XXVIII.* Springer, 276–291.
[77] Zhongyuan Xu and Scott D Stoller. 2015. Mining attribute-based access control policies. *IEEE Transactions on Dependable and Secure Computing* 12, 5 (2015), 533–545.
[78] Kan Yang, Zhen Liu, Xiaohua Jia, and Xuemin Sherman Shen. 2016. Time-domain attribute-based access control for cloud-based video content sharing: A cryptographic approach. *IEEE Transactions on Multimedia* 18, 5 (2016), 940–950.
[79] Qiang Yang, Haining Henry Zhang, and Tianyi Li. 2001. Mining web logs for prediction models in WWW caching and prefetching. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 473–478.
[80] Dana Zhang, Kotagiri Ramamohanarao, and Tim Ebringer. 2007. Role engineering using graph optimisation. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies.* ACM, 139–144.

A SIMPLIFYING THE COMPUTATION OF EXPECTATIONS

We prove here Lemma 2. We start with some auxiliary lemmas and definitions.

Lemma 3. *Let $\mathfrak{f}$ and $\mathfrak{g}$ be facts, φ be a formula in $\mathcal{L}$, $(\mathrm{u}, \mathrm{p}) \in U \times P$, and $\{\psi_i\}_i \subseteq \mathcal{L}$ such that $\{\psi_i^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})\}_i$ is a set of mutually independent random variables under the distribution q.*

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[\mathfrak{g}] = \begin{cases} \mathrm{b} & \text{if } \mathfrak{f} = \mathfrak{g} \\ \sum_{\mathrm{b}\in Range(\mathfrak{g})} \tilde{q}_{\mathfrak{g}}(\mathrm{b})\,\mathrm{b} & \text{otherwise.} \end{cases}$$

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[(\neg\varphi)^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})\right] = 1 - \mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\varphi^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})\right].$$

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\left(\bigwedge_i \psi_i\right)^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})\right] = \prod_i \mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\psi_i^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})\right].$$

Proof. Observe that, for a Bernoulli random variable X, $\mathbb{E}[X] = \mathbb{P}(X = 1)$. Recall also that $\mathbb{E}[XY] = \mathbb{E}[X]\,\mathbb{E}[Y]$, whenever X and Y are mutually independent. With these observations and using standard probability laws, one can derive the equations above. □

Lemma 4. *Let $\varphi \in \mathcal{L}$ and let $(\mathrm{u}, \mathrm{p}) \in U \times P$. If $\varphi^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})$ is diverse, then $\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\varphi^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})\right]$ can be computed using only the equations from Lemma 2.*

This lemma is proved by induction on φ and by recalling that any two different random facts are independent under the distribution q, which follows from the way that the distribution q is factorized.

Corollary 2.

$$\mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}[L(Auth, \mathfrak{X}; \varphi)] = \sum_{(\mathrm{u},\mathrm{p})\in U\times P} \left| Auth(\mathrm{u}, \mathrm{p}) - \mathbb{E}_{\mathfrak{f}\mapsto\mathrm{b}}\left[\varphi^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})\right]\right|.$$

Proof. $L(Auth, \mathfrak{X}; \varphi)$ can be rewritten as follows:

$$\sum_{(\mathrm{u},\mathrm{p})\in Auth} \left(1 - \varphi^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})\right) + \sum_{(\mathrm{u},\mathrm{p})\in U\times P\setminus Auth} \varphi^{\mathfrak{X}}(\mathrm{u}, \mathrm{p}).$$

The result follows from the linearity of expectation. □

Lemma 2 follows from Lemma 4 and Corollary 2.

B POLICY MINERS BUILT USING UNICORN

We present here technical details on how we built policy miners for different policy languages using Unicorn.

B.1 ABAC policies

ABAC is an access control paradigm where permissions are assigned to users depending on the users' and the permission's *attribute values.* An *ABAC policy* is a set of *rules.* A rule is a set of attribute values. Recall that a request (u, p) is a pair consisting of a user $u \in U$ and a permission $p \in P$. A rule *assigns* a permission p to a user u if u and p's permission attribute values contain all of the rule's attribute values. A policy *assigns* p to u if some rule in the policy assigns p to u.

When mining ABAC policies, we are not only given a permission assignment $Auth \subseteq U \times P$, but also *attribute assignment relations* $UAtt \subseteq U \times AttVals$ and $PAtt \subseteq P \times AttVals$ that describe what attribute values each user and each permission has. Here, $AttVals$ denotes the set of possible attribute values. We refer to previous work for a discussion on how to obtain these attribute assignment relations [18, 77].

The objective in mining ABAC policies is to find a set of rules that assigns permissions to users based on the users' and the permissions' attribute values. We explain next how to build a policy miner for ABAC using Unicorn. Let $Rules$ and $AttVals$ be sorts for rules and attribute values, respectively. Let RUA and RPA be flexible binary relation symbols of type $Rules \times AttVals$. For $M, N \in \mathbb{N}$, the formula $\varphi^{ABAC}_{M,N}(u, p)$ below is a template formula for ABAC:

$$\bigvee_{i \leq N} \bigwedge_{j \leq M} \left(\begin{array}{l} RUA\left(\underline{s}_i, \underline{a}_j\right) \rightarrow \underline{UAtt}(u, \underline{a}_j)\right) \wedge \\ RPA\left(\underline{s}_i, \underline{a}_j\right) \rightarrow \underline{PAtt}(p, \underline{a}_j)\right) \end{array} \right). \qquad (14)$$

In this formula, $\underline{s}_i$, for $i \leq N$, is a rigid constant symbol of sort $Rules$ denoting a rule. The symbol $\underline{a}_j$, for $j \leq M$, is a rigid constant denoting an attribute value. The formula $RUA\left(\underline{s}_i, \underline{a}_j\right)$ describes whether rule $\underline{s}_i$ requires the user to have the attribute value $\underline{a}_j$. The formula $RPA\left(\underline{s}_i, \underline{a}_j\right)$ describes an analogous requirement. We use two rigid relation symbols $\underline{UAtt}$ and $\underline{PAtt}$ to represent the attribute assignment relations. The formulas $\underline{UAtt}\left(u, \underline{a}_j\right)$ and $\underline{PAtt}\left(p, \underline{a}_j\right)$ describe whether u and p, respectively, are assigned the attribute value $\underline{a}_j$. Intuitively, the formula $\varphi^{ABAC}_{M,N}(u, p)$ is satisfied by $(\mathrm{u}, \mathrm{p}) \in U \times P$ if, for some rule $\underline{s}_i$, (u, p) possesses all user and permission attribute values required by $\underline{s}_i$ under RUA and RPA.

Observe that a policy miner does not need to find an interpretation for the symbols $\underline{UAtt}$ and $\underline{PAtt}$ because *the organization already has interpretations for those symbols*. When mining ABAC policies, the organization already knows what attribute values each user and each permission has and wants to mine from them an ABAC policy. The miner only needs to specify which attribute values must be required by each rule. This is why we specify the attribute assignment relations with rigid symbols.

We use $L(Auth, \mathfrak{I}; \varphi^{ABAC}_{M,N})$ as the objective function. Observe that every atomic formula occurs at most once in $\varphi^{ABAC}_{M,N}$, so, by Corollary 1, we can use Lemma 2 to compute all relevant expectations.

Finally, we can also add a regularization term to $L(Auth, \mathfrak{I}; \varphi^{ABAC}_{M,N})$ to avoid mining policies with too many rules or unnecessarily large rules. One such regularization term is

$$\|\mathfrak{I}\| = \sum_{i \leq N} \sum_{j \leq M} RUA^{\mathfrak{I}}\left(\underline{s}_i, \underline{a}_j\right) + RPA^{\mathfrak{I}}\left(\underline{s}_i, \underline{a}_j\right).$$

The expression $\|\mathfrak{I}\|$ counts the number of attribute values required by each rule, which is a common way to measure an ABAC policy's complexity [18, 77]. If we instead use the objective function $\lambda \|\mathfrak{I}\| + L(Auth, \mathfrak{I}; \varphi^{ABAC}_{M,N})$, then the objective function penalizes not only policies that differ substantially from $Auth$, but also policies that are too complex. Observe that $\|\mathfrak{X}\|$ is diverse. Hence, we can use the linearity of expectation and Lemma 2 to compute all expectations needed to implement Algorithm 1.

B.2 ABAC policies from logs

Some miners are geared towards mining policies from logs of access requests [18, 56, 76]. We now present an objective function that can be used to mine ABAC policies from access logs, instead of permission assignments. We let $\varphi := \varphi^{ABAC}_{M,N}$ for the rest of this subsection.

A log G is a disjoint union of two subsets A and D of $U \times P$, denoting the requests that have been *authorized* and *denied*, respectively.

In the case of ABAC, a policy mined from a log should aim to fulfill three requirements. The policy should be *succinct*, *generalize well*, and be *precise* [18]. Therefore, we define an objective function $L'_{ABAC}(G, \mathfrak{I})$ as the sum

$$L'_{ABAC}(G, \mathfrak{I}) = \lambda_0 \|\mathfrak{I}\| + L_1(G, \mathfrak{I}) + L_2(G, \mathfrak{I}). \qquad (15)$$

The term $\|\mathfrak{I}\|$ is as defined in Section B.1 and aims to make the policy succinct by penalizing complex policies. The term $L_1(G, \mathfrak{I})$ aims to make the mined policy generalize well and is defined as

$$L_1(G, \mathfrak{I}) = \lambda_{1,1} \sum_{(\mathrm{u},\mathrm{p}) \in A} \left(1 - \varphi^{\mathfrak{I}}(\mathrm{u}, \mathrm{p})\right) + \lambda_{1,2} \sum_{(\mathrm{u},\mathrm{p}) \in D} \varphi^{\mathfrak{I}}(\mathrm{u}, \mathrm{p}).$$

Finally, the function $L_2(G, \mathfrak{I})$ aims to make the mined policy precise by penalizing policies that authorize too many requests that are not in the log.

$$L_2(G, \mathfrak{I}) = \lambda_2 \sum_{(\mathrm{u},\mathrm{p}) \in U \times P \setminus G} \varphi^{\mathfrak{I}}(\mathrm{u}, \mathrm{p}).$$

One can show that $\varphi^{\mathfrak{X}}(\mathrm{u}, \mathrm{p})$ is diverse, for any $(\mathrm{u}, \mathrm{p}) \in U \times P$. Therefore, we can compute $\mathbb{E}_{\mathfrak{f} \mapsto \mathrm{b}}\left[L'_{ABAC}(G, \mathfrak{X})\right]$ using only the linearity of expectation and Lemma 2.

B.3 Business-meaningful RBAC policies

Frank et. al. [30] developed a probabilistic policy miner for RBAC policies that incorporated business information. Aside from a permission assignment, the miner takes as input an *attribute-assignment relation* $AA \subseteq U \times AVal$, where $AVal$ denotes all possible combination of attribute values. It is assumed that each user is assigned exactly one combination of attribute values.

This miner grants similar sets of roles to users that have similar attribute values. For this, it uses the following formula $\Delta(\mathrm{u}, \mathrm{u}', \mathfrak{I})$ that measures the disagreement between the roles that a policy $\mathfrak{I}$ assigns to two users u and u′:

$$\Delta(\mathrm{u}, \mathrm{u}', \mathfrak{I}) = \sum_{i \leq N} UA^{\mathfrak{I}}(\mathrm{u}, \underline{r}_i)\left(1 - 2UA^{\mathfrak{I}}(\mathrm{u}, \underline{r}_i) UA^{\mathfrak{I}}(\mathrm{u}', \underline{r}_i)\right).$$

The formula $\|\mathfrak{I}\|$ below shows how Frank et al.'s miner measures an RBAC policy's complexity. The complexity increases whenever two users with the same combination of attribute values get assigned significantly different sets of roles.

$$\|\mathfrak{I}\| = \frac{1}{N} \sum_{\mathrm{u},\mathrm{u}' \in U} \sum_{\underline{\mathrm{a}} \in AVal} \underline{AA}(\mathrm{u}, \underline{\mathrm{a}}) \underline{AA}(\mathrm{u}', \underline{\mathrm{a}}) \Delta(\mathrm{u}, \mathrm{u}', \mathfrak{I}).$$

Here, N denotes the total of users. Note that $\underline{AA}$ is a rigid relation symbol representing AA. Its interpretation is therefore fixed and not computed by the policy miner.

To mine business-meaningful RBAC policies, we use the objective function $\lambda \|\mathfrak{I}\| + L\left(Auth, \mathfrak{I}; \varphi_N^{RBAC}\right)$, where $\lambda > 0$ is a trade-off hyper-parameter. Observe that this objective function penalizes the following types of policies.

- Policies that assign significantly different sets of roles to users with the same attribute values.
- Policies whose assignment of permissions to users substantially differs from the assignment given by *Auth*.

The random variable $\|\mathfrak{X}\|$ is, however, not diverse. This is because, for $i \leq N$, the random fact $UA^{\mathfrak{X}}(\mathsf{u}, \underline{\mathsf{r}}_i)$ occurs more than once in $\Delta(\mathsf{u}, \mathsf{u}', \mathfrak{X})$. Nonetheless, observe that

$$\Delta(\mathsf{u}, \mathsf{u}', \mathfrak{X}) = \sum_{i \leq N} UA^{\mathfrak{X}}(\mathsf{u}, \underline{\mathsf{r}}_i) - 2\left(UA^{\mathfrak{X}}(\mathsf{u}, \underline{\mathsf{r}}_i)\right)^2 UA^{\mathfrak{X}}(\mathsf{u}', \underline{\mathsf{r}}_i).$$

One can then compute $\mathbb{E}_{\mathfrak{f}\to\mathfrak{b}}[\Delta(\mathsf{u}, \mathsf{u}', \mathfrak{X})]$ by using the linearity of expectation and the fact that $\mathbb{E}[X^n] = (\mathbb{E}[X])^n$, for $n \in \mathbb{N}$ and X a Bernoulli random variable. Hence,

$$\mathbb{E}_{\mathfrak{f}\to\mathfrak{b}}[\Delta(\mathsf{u}, \mathsf{u}', \mathfrak{X})] = \sum_{i \leq N} \left(\begin{array}{l} \mathbb{E}_{\mathfrak{f}\to\mathfrak{b}}\left[UA^{\mathfrak{X}}(\mathsf{u}, \underline{\mathsf{r}}_i)\right] - \\ 2\left(\mathbb{E}_{\mathfrak{f}\to\mathfrak{b}}\left[UA^{\mathfrak{X}}(\mathsf{u}, \underline{\mathsf{r}}_i)\right]\right)^2 \mathbb{E}_{\mathfrak{f}\to\mathfrak{b}}\left[UA^{\mathfrak{X}}(\mathsf{u}', \underline{\mathsf{r}}_i)\right] \end{array} \right).$$

One can check that this observation and Lemma 2 suffice to compute the expectations necessary for Algorithm 1.

C MINING XACML POLICIES

Although XACML is the de facto standard for access control specification, no algorithm has previously been proposed for mining XACML policies. We now illustrate how, using UNICORN, we have built the first XACML policy miner.

C.1 Background

XACML syntax. To simplify the presentation, we use a reduced version of XACML, given as a BNF grammar below. However, our approach extends to the core XACML. Moreover, our reduced XACML is still powerful enough to express Continue [27, 47], a benchmark XACML policy used for policy analysis.

$$\begin{array}{rcl} \langle\texttt{Dec}\rangle & ::= & \textit{allow} \mid \textit{deny} \\ \langle\texttt{Rule}\rangle & ::= & (\langle\texttt{Dec}\rangle, \alpha) \\ \langle\texttt{Comb}\rangle & ::= & \textit{FirstApp} \mid \textit{AllowOv} \mid \textit{DenyOv} \\ \langle\texttt{Pol}\rangle & ::= & (\langle\texttt{Comb}\rangle, (\langle\texttt{Pol}\rangle^* \mid \langle\texttt{Rule}\rangle^*)) \end{array}$$

Fix a set *AVals* of attribute values. An *XACML rule* is a pair (δ, α), where $\delta \in \{allow, deny\}$ is the *rule's decision* and α is a subset of *AVals*. An XACML *policy* is a pair $(\kappa, \bar{\pi})$, where $\kappa \in \{FirstApp, AllowOv, DenyOv\}$ is a *combination algorithm* and $\bar{\pi}$ is either a list of policies or a list of XACML rules. *FirstApp*, *AllowOv*, *DenyOv* denote XACML's standard policy combination algorithms. We explain later how they work. For a policy π, we denote its combination algorithm by $Comb(\pi)$ and, for a rule r, we denote its decision by $Dec(r)$.

XACML semantics. We now recall XACML's semantics. A *request* is a subset of *AVals* denoting the attribute values that a subject s, an action a, and an object o satisfy when s attempts to execute a on o. We denote by 2^{AVals} the set of requests. A request *satisfies* a rule (δ, α) if the request contains all attributes in α. In this case, if $\delta = allow$, then we say that the rule *authorizes the request*; otherwise, we say that the rule *denies the request.*

A policy π of the form $(AllowOv, \left(\pi_1', \ldots, \pi_\ell'\right))$ authorizes a request z if there is an $i \leq \ell$ such that π_i' authorizes z. The policy π *denies* z if no π_i', for $i \leq \ell$, authorizes z, but some π_j', for $j \leq \ell$, denies it.

A policy π of the form $(DenyOv, \left(\pi_1', \ldots, \pi_\ell'\right))$ denies a request if some π_i' denies it. The policy authorizes the request if no π_i' denies it, but some π_i' authorizes it.

A policy π of the form $(FirstApp, \left(\pi_1', \ldots, \pi_\ell'\right))$ authorizes a request if there is an $i \leq \ell$ such that π_i' authorizes it and π_j', for $j < i$, neither authorizes it nor denies it. The policy denies a request if there is $i \leq \ell$ such that π_i' denies it and π_j', for $j < i$, neither authorizes it nor denies it.

C.2 Auxiliary definitions

For a policy $\pi = \left(\kappa, \left(\pi_1', \ldots, \pi_\ell'\right)\right)$, we call π_i' a *child* of π. A policy is a *descendant* of π if it is a child of π or is a descendant of a child of π.

A policy π has *breadth* $N \in \mathbb{N}$ if $\ell \leq N$ and each of π's children is either a rule or has breadth N. A policy π has *depth* is $M \in \mathbb{N}$ (i) if $M = 1$ and each of its children is a rule, or (ii) if $M > 1$ and some child of π has depth $M - 1$ and the rest have depth at most $M - 1$.

Two formulas $\psi_1, \psi_2 \in \mathcal{L}$ are *mutually exclusive* if there is no $\mathfrak{I}$ and no $\mathsf{z} \in 2^{AVals}$ such that both $\psi_1^{\mathfrak{I}}(\mathsf{z})$ and $\psi_2^{\mathfrak{I}}(\mathsf{z})$ hold. When ψ_1 and ψ_2 are mutually exclusive, we write $\psi_1 \oplus \psi_2$ instead of $\psi_1 \vee \psi_2$.

C.3 A template formula for XACML

For $M, N \in \mathbb{N}$, we present a template formula for the language of all XACML policies of depth and breadth at most M and N, respectively.

Let $\mathcal{S}$ be the set of all N-ary sequences of length at most M and let $\epsilon \in \mathcal{S}$ be the empty sequence. For $j \in \{0, \ldots, N-1\}$, we denote by $\sigma \triangleright j$ the result of appending j to σ and by $j \triangleleft \sigma$ the result of prepending j to σ.

Let **REQS** be a sort representing all requests, **AVALS** be a sort representing all attribute values, and **POLS** a sort representing policies and rules. For each $\sigma \in \mathcal{S}$, define a rigid constant $\underline{\mathsf{y}}_\sigma$ symbol of sort **POLS** such that $\underline{\mathsf{y}}_\sigma \neq \underline{\mathsf{y}}_{\sigma'}$, whenever $\sigma \neq \sigma'$.

The set of rigid constants $\{\underline{\mathsf{y}}_\sigma \mid \sigma \in \mathcal{S}\}$ are intended to represent a tree of XACML policies and rules. The constant $\underline{\mathsf{y}}_\epsilon$ is the root policy. For $\sigma \in \mathcal{S}$ with length less than M and $j \in \{0, \ldots, N-1\}$, the constant $\underline{\mathsf{y}}_{\sigma \triangleright j}$ represents one of $\underline{\mathsf{y}}_\sigma$'s children.

Let z be a variable of sort **REQS**. Formula C.1 presents a template formula for the XACML fragment introduced above. We explain its main parts.

We define signature symbols that represent all terminal symbols in the BNF grammar above. For example, we define two rigid constant symbols `XAllows` and `XDenies` that represent the decisions *allow* and *deny*. We define two flexible function symbols *XDec* and *XComb*. For a rigid constant $\underline{\mathsf{y}}_\sigma$, $XDec\left(\underline{\mathsf{y}}_\sigma\right)$ denotes the decision

$$
\begin{aligned}
\varphi^{\mathrm{XACML}}_{M,N}\left(\underline{\mathsf{y}}_{\epsilon}, z\right) &:= \texttt{allows}\left(\underline{\mathsf{y}}_{\epsilon}, z\right).\\
\texttt{allows}\left(\underline{\mathsf{y}}_{\sigma}, z\right) &:= \left(\mathit{XIsRule}\left(\underline{\mathsf{y}}_{\sigma}\right) \rightarrow \texttt{allowsRule}\left(\underline{\mathsf{y}}_{\sigma}, z\right)\right) \wedge \\
&\qquad \left(\neg \mathit{XIsRule}\left(\underline{\mathsf{y}}_{\sigma}\right) \rightarrow \texttt{allowsPol}\left(\underline{\mathsf{y}}_{\sigma}, z\right)\right).\\
\texttt{allowsRule}\left(\underline{\mathsf{y}}_{\sigma}, z\right) &:= \mathit{XActive}\left(\underline{\mathsf{y}}_{\sigma}\right) \wedge \mathit{XDec}\left(\underline{\mathsf{y}}_{\sigma}\right) = \underline{\mathit{allow}} \wedge z \vDash \underline{\mathsf{y}}_{\sigma}.\\
\texttt{allowsPol}\left(\underline{\mathsf{y}}_{\sigma}, z\right) &:= \mathit{XActive}\left(\underline{\mathsf{y}}_{\sigma}\right) \wedge \\
&\left[\begin{array}{l}
\left(\begin{array}{l}\mathit{XComb}\left(\underline{\mathsf{y}}_{\sigma}\right) = \underline{\mathit{AllowOv}} \wedge \\ \bigvee_{j \leq N} \texttt{allows}\left(\underline{\mathsf{y}}_{\sigma\triangleright j}, z\right)\end{array}\right) \oplus \\
\left(\begin{array}{l}\mathit{XComb}\left(\underline{\mathsf{y}}_{\sigma}\right) = \underline{\mathit{FirstApp}} \wedge \\ \bigoplus_{j \leq N}\left(\begin{array}{l}\bigwedge_{i<j} \texttt{NA}\left(\underline{\mathsf{y}}_{\sigma\triangleright i}, z\right) \wedge \\ \texttt{allows}\left(\underline{\mathsf{y}}_{\sigma\triangleright j}, z\right)\end{array}\right)\end{array}\right) \oplus \\
\left(\begin{array}{l}\mathit{XComb}\left(\underline{\mathsf{y}}_{\sigma}\right) = \underline{\mathit{DenyOv}} \wedge \\ \bigoplus_{j \leq N}\left(\begin{array}{l}\bigwedge_{i<j} \texttt{NA}\left(\underline{\mathsf{y}}_{\sigma\triangleright i}, z\right) \wedge \\ \texttt{allows}\left(\underline{\mathsf{y}}_{\sigma\triangleright j}, z\right) \wedge \\ \bigwedge_{i<k} \neg\texttt{denies}\left(\underline{\mathsf{y}}_{\sigma\triangleright k}, z\right)\end{array}\right)\end{array}\right)
\end{array}\right].
\end{aligned}
$$

Formula C.1: A template formula for XACML

of the rule represented by $\underline{\mathsf{y}}_{\sigma}$. Similarly, $\mathit{XComb}\left(\underline{\mathsf{y}}_{\sigma}\right)$ denotes the combination algorithm of the policy represented by $\underline{\mathsf{y}}_{\sigma}$.

The formula $\texttt{allows}\left(\underline{\mathsf{y}}_{\sigma}, z\right)$ holds if $\underline{\mathsf{y}}_{\sigma}$ authorizes the request represented by z. The formula $\texttt{denies}\left(\underline{\mathsf{y}}_{\sigma}, z\right)$ holds if $\underline{\mathsf{y}}_{\sigma}$ denies the request represented by z and is defined analogously. Observe that $\texttt{allows}\left(\underline{\mathsf{y}}_{\sigma}, z\right)$ and $\texttt{denies}\left(\underline{\mathsf{y}}_{\sigma}, z\right)$ denote formulas. Hence, allows and denies are not symbols in the signature we use to specify $\varphi^{\mathrm{XACML}}_{M,N}$.

XActive is a flexible relation symbol and $\mathit{XActive}\left(\underline{\mathsf{y}}_{\sigma}\right)$ holds if $\underline{\mathsf{y}}_{\sigma}$ is a descendant of $\underline{\mathsf{y}}_{\epsilon}$.

The formula $\texttt{NA}\left(\underline{\mathsf{y}}_{\sigma}, z\right)$ holds if $\underline{\mathsf{y}}_{\sigma}$ neither authorizes nor denies the request represented by z. It can be expressed in $\mathcal{L}$ as follows:

$$\texttt{NA}\left(\underline{\mathsf{y}}_{\sigma}, z\right) := \neg \mathit{XActive}\left(\underline{\mathsf{y}}_{\sigma}\right) \vee \bigwedge_{j \leq N} \texttt{NA}\left(\underline{\mathsf{y}}_{\sigma\triangleright j}, z\right).$$

The formula $z \vDash \underline{\mathsf{y}}_{\sigma}$ holds if all attributes required by $\underline{\mathsf{y}}_{\sigma}$ are contained by the request represented by z. This formula can be expressed in $\mathcal{L}$ as follows:

$$\bigwedge_{\mathsf{a} \in \mathit{AVals}} \left(\mathit{XRequiresAVal}\left(\underline{\mathsf{y}}_{\sigma}, \underline{\mathsf{a}}\right) \rightarrow \underline{\mathit{hasAttVal}}\left(z, \underline{\mathsf{a}}\right)\right),$$

where *XRequiresAVal* is a flexible relation symbol and $\underline{\mathit{hasAttVal}}$ and $\underline{\mathsf{a}}$, for $\mathsf{a} \in \mathit{AVals}$, are rigid symbols. For a policy $\mathfrak{I}$, $\mathit{XRequiresAVal}^{\mathfrak{I}}\left(\underline{\mathsf{y}}_{\sigma}, \underline{\mathsf{a}}\right)$ holds if $\underline{\mathsf{y}}_{\sigma}$ is a rule and requires attribute a to be satisfied. The formula $\underline{\mathit{hasAttVal}}\left(z, \underline{\mathsf{a}}\right)$ checks if the request contains attribute a.

In the full version, we show that the formula $\varphi^{\mathrm{XACML}}_{M,N}$ is a template formula for the language of all XACML policies of depth and breadth at most M and N, respectively.

Having a template formula for this XACML fragment, we now define an objective function. An example of an objective function is $\lambda \left\|\mathfrak{I}\right\| + L\left(\mathit{Auth}, \mathfrak{I}; \varphi^{\mathrm{XACML}}_{M,N}\right)$, where $\lambda > 0$ is a hyper-parameter and $\left\|\mathfrak{I}\right\|$ defines $\mathfrak{I}$'s complexity. We inductively define the complexity $\mathit{compl}\left(\pi\right)$ of a XACML policy π as follows.

- If $\pi = (\delta, \alpha)$, then $\mathit{compl}\left(\pi\right) = |\alpha|$.
- If $\pi = (\kappa, (\pi_1, \ldots, \pi_k))$, then $\mathit{compl}\left(\pi\right) = |\alpha|$.

Finally, we define $\left\|\mathfrak{I}\right\|$ as $\mathit{compl}\left(\mathcal{M}\left(\mathfrak{I}\right)\right)$.

C.4 Computing expectations

For a formula $\varphi \in \mathcal{L}$ and a request $\mathsf{z} \in 2^{\mathit{AVals}}$, we define the random variable $\varphi^{\mathfrak{X}}\left(\mathsf{z}\right)$ in a way similar to the one given in Section 6.1. We now give some auxiliary definitions and lemmas that help to compute $\mathbb{E}_{\mathfrak{f}\mapsto \mathsf{b}}\left[\left(\varphi^{\mathrm{XACML}}_{M,N}\right)^{\mathfrak{X}}\left(\mathsf{z}\right)\right]$. Proofs are in the full version.

Lemma 5. Let $\mathsf{z} \in 2^{\mathit{AVals}}$ and ψ_1, ψ_2 be mutually exclusive formulas, then

$$\mathbb{E}_{\mathfrak{f}\mapsto \mathsf{b}}\left[\left(\psi_1 \oplus \psi_2\right)^{\mathfrak{X}}\left(\mathsf{z}\right)\right] = \mathbb{E}_{\mathfrak{f}\mapsto \mathsf{b}}\left[\psi_1^{\mathfrak{X}}\left(\mathsf{z}\right)\right] + \mathbb{E}_{\mathfrak{f}\mapsto \mathsf{b}}\left[\psi_2^{\mathfrak{X}}\left(\mathsf{z}\right)\right].$$

Definition 12. A set $\Phi \subseteq \mathcal{L}$ of formulas is *unrelated* if for every $\varphi \in \Phi$ and every atomic formula α occurring in φ, there is no $\psi \in \Phi \setminus \{\varphi\}$ such that α occurs in φ. □

Lemma 6. If $\mathsf{z} \in 2^{\mathit{AVals}}$ and Φ is a set of unrelated formulas, then $\left\{\varphi^{\mathfrak{X}}\left(\mathsf{z}\right) \mid \varphi \in \Phi\right\}$, under the distribution q, is mutually independent.

Lemma 7. We can compute $\mathbb{E}_{\mathfrak{f}\mapsto \mathsf{b}}\left[\left(\varphi^{\mathrm{XACML}}_{M,N}\right)^{\mathfrak{X}}\left(\mathsf{z}\right)\right]$ using only the equations given in Lemmas 3 and 5.

Proof. Observe that every atomic formula in $\texttt{allowsRule}\left(\underline{\mathsf{y}}_{\sigma}, z\right)$ occurs exactly once, so $\texttt{allowsRule}^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma}, \mathsf{z}\right)$ is diverse. Hence, by Corollary 1, we can use Lemma 2 to compute the expectation $\mathbb{E}_{\mathfrak{f}\mapsto \mathsf{b}}\left[\texttt{allowsRule}^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma}, \mathsf{z}\right)\right]$.

The formula $\texttt{allowsPol}\left(\underline{\mathsf{y}}_{\sigma}, z\right)$ can be rewritten as

$$\left(\mathit{XActive}\left(\underline{\mathsf{y}}_{\sigma}\right) \wedge \psi_1\left(\underline{\mathsf{y}}_{\sigma}, z\right)\right) \oplus \left(\mathit{XActive}\left(\underline{\mathsf{y}}_{\sigma}\right) \wedge \psi_2\left(\underline{\mathsf{y}}_{\sigma}, z\right)\right) \oplus \left(\mathit{XActive}\left(\underline{\mathsf{y}}_{\sigma}\right) \wedge \psi_3\left(\underline{\mathsf{y}}_{\sigma}, z\right)\right).$$

Each formula $\psi_i\left(\underline{\mathsf{y}}_{\sigma}, z\right)$ is built from a set of unrelated formulas. Hence, by Lemma 6, we can use Lemma 3 to compute $\mathbb{E}_{\mathfrak{f}\mapsto \mathsf{b}}\left[\psi_i\left(\underline{\mathsf{y}}_{\sigma}, z\right)\right]$.

Using Lemmas 5 and 3, we can show that

$$\mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\texttt{allowsPol}^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma},\mathsf{z}\right)\right] =$$

$$\mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\mathit{XActive}^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma},\mathsf{z}\right)\right] \times \left(\begin{array}{l} \mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\psi_1^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma},\mathsf{z}\right)\right] + \\ \mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\psi_2^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma},\mathsf{z}\right)\right] + \\ \mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\psi_3^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma},\mathsf{z}\right)\right] \end{array}\right).$$

Therefore, $\mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\texttt{allowsPol}^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma},\mathsf{z}\right)\right]$ can be computed using only Lemmas 5 and 3.

Finally, recall that $\varphi_{M,N}^{\text{XACML}}\left(\underline{\mathsf{y}}_{\epsilon},z\right) = \texttt{allows}\left(\underline{\mathsf{y}}_{\epsilon},z\right)$. Observe now that $\texttt{allows}\left(\underline{\mathsf{y}}_{\epsilon},z\right)$ is built from the following unrelated set:

$$\left\{\mathit{XIsRule}\left(\underline{\mathsf{y}}_{\sigma}\right), \texttt{allowsRule}\left(\underline{\mathsf{y}}_{\sigma},z\right), \texttt{allowsPol}\left(\underline{\mathsf{y}}_{\sigma},z\right)\right\}.$$

By Lemma 6, the corresponding set of random variables is independent. Hence, we can use Lemma 3 to reduce the computation of $\mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\left(\varphi_{M,N}^{\text{XACML}}\right)^{\mathfrak{X}}(\mathsf{z})\right]$ to the computation of $\mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\mathit{XIsRule}^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma}\right)\right]$, $\mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\texttt{allowsRule}^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma},\mathsf{z}\right)\right]$, and $\mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\texttt{allowsPol}^{\mathfrak{X}}\left(\underline{\mathsf{y}}_{\sigma},\mathsf{z}\right)\right]$. However, as observed above, all these expectations can be computed using Lemmas 3 and 5. Hence, we can compute $\mathbb{E}_{\mathsf{f}\to\mathsf{b}}\left[\left(\varphi_{M,N}^{\text{XACML}}\right)^{\mathfrak{X}}(\mathsf{z})\right]$ using only those two lemmas. □

Having proven the previous lemmas, we can now implement Algorithm 1 to produce a policy miner for XACML policies.

D DATASETS AND SYNTHETIC POLICIES USED FOR EXPERIMENTS

D.1 Datasets for BM-RBAC

We use the access control matrix provided by Amazon for the IEEE MLSP 2012 competition [38]. They assign three types of permissions, named "HOST", "PERM_GROUP", and "SYSTEM_GROUP". For each type of permission, we created an access control matrix by collecting all users and all permissions belonging to that type. There are approximately 30,000 users, 1,700 permissions of type "HOST", 6,000 of type "PERM_GROUP", and 20,000 of type "SYSTEM_GROUP".

The resulting access control matrices are far too large to be handled efficiently by the policy miner we developed. To address this, during 5-fold cross-validation, we worked instead with an access control submatrix induced by a sample of 30% of all users. Each fold used a different sample of users. To see why this is enough, we remark that, in RBAC policies, the number R of roles is usually much smaller than the number N of users. Moreover, the number K of possible subsets of permissions that users are assigned by RBAC policies is small in comparison to the whole set of possible subsets of permissions. If N is much larger than K, then, by the pidgeonhole principle, many users have the same subset of permissions. Therefore, it is not necessary to use all N users to mine an adequate RBAC policy, as only a fraction of them has all the necessary information. The high TPR (above 80%) of the policy that we mined supports the fact that using a submatrix is still enough to mine policies that generalize well.

D.2 Synthetic policy for spatio-temporal RBAC

We present here the synthetic spatio-temporal RBAC policy that we used for our experiments. We assume the existence of five rectangular buildings, described in Table 10. The left column indicates the building's name and the right column describes the two-dimensional coordinates of the building's corners. There are five roles, which we describe next. We regard a permission as an *action* executed on an *object*.

Name	Corners
Main building	(1, 3), (1, 4), (4, 4), (4, 3)
Library	(1, 1), (1, 2), (2, 2), (2, 1)
Station	(8, 1), (8, 9), (9, 9), (9, 1)
Laboratory	(2, 6), (2, 8), (4, 8), (4, 6)
Computer room	(6, 6), (6, 7), (7, 7), (7, 6)

Table 10

The first role assigns a permission to a user if all of the following hold:

- The user is at most 1 meter away from the computer room.
- The object is in the computer room or in the laboratory.
- The current day is an odd day of the month.
- The current time is between 8AM and 5PM.

The second role assigns a permission to a user if all of the following hold:

- The user is outside the library.
- The object is at most 1 meter away from the library.
- Either
  - the current day is before the 10th day of the month and the current time is between 2PM and 8PM or
  - the current day is after the 15th day of the month and the current time is between 8AM and 12PM.

The third role assigns a permission to a user if all of the following hold:

- The user is at most 3 meters away from the main building.
- The object is at most 3 meters away from the main building.

The fourth role assigns a permission to a user if all of the following hold:

- The user is inside the library.
- The object is outside the library.
- The current day is before the 15th day of the month.
- The current time is between 12AM and 12PM.

The fifth role assigns a permission to a user if all of the following hold:

- The user is inside the main building, at most 1 meter away from the library, inside the laboratory, at most 2 meters away from the computer room, or inside the station.
- The object satisfies the same spatial constraint.
- The current day is before the 15th day of the month.
- The current time is between 12AM and 12PM.