Methods and Tools for Policy Analysis

AMANI ABU JABAL, MARYAM DAVARI, and ELISA BERTINO, Purdue University, USA
CHRISTIAN MAKAYA, SERAPHIN CALO, and DINESH VERMA, IBM Research, USA
ALESSANDRA RUSSO, Imperial College, UK
CHRISTOPHER WILLIAMS, The Defence Science and Technology Laboratory, UK

Policy-based management of computer systems, computer networks and devices is a critical technology especially for present and future systems characterized by large-scale systems with autonomous devices, such as robots and drones. Maintaining reliable policy systems requires efficient and effective analysis approaches to ensure that the policies verify critical properties, such as correctness and consistency. In this paper, we present an extensive overview of methods for policy analysis. Then, we survey policy analysis systems and frameworks that have been proposed and compare them under various dimensions. We conclude the paper by outlining novel research directions in the area of policy analysis.

CCS Concepts: • **Security and privacy → Access control**; **Authorization**; **Distributed systems security**; **Firewalls**; **Network security**;

Additional Key Words and Phrases: Policy analysis, access control policies, network policies, policy quality requirements, policy design and organization, similarity analysis, change impact analysis

ACM Reference format:
Amani Abu Jabal, Maryam Davari, Elisa Bertino, Christian Makaya, Seraphin Calo, Dinesh Verma, Alessandra Russo, and Christopher Williams. 2019. Methods and Tools for Policy Analysis. *ACM Comput. Surv.* 51, 6, Article 121 (February 2019), 35 pages.
https://doi.org/10.1145/3295749

1 INTRODUCTION

Advances in robotics and artificial intelligence plus the explosion of low-cost mobile phones, drones, embedded devices, and the Internet of Things (IoT) are leading to the massive deployment of autonomous systems. These systems will include intelligent autonomous devices able to collaborate, to gather fine-grained information from the operating environments, and to physically act in these environments. Many such devices will have very rich cognitive capabilities and be able to control other devices autonomously and will need to interact with other autonomous

This research was sponsored by the U.S. Army Research Laboratory and the U.K. Ministry of Defence under Agreement Number W911NF-16-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.
Authors' addresses: A. Abu Jabal, M. Davari, and E. Bertino, Purdue University, 305 N. University Street, West Lafayette, IN, 47907, USA; emails: {aabujaba, davari, bertino}@purdue.edu; C. Makaya, S. Calo, and D. Verma, IBM Research, Yorktown Heights, NY, USA; emails: wchrismak@gmail.com, {scalo, dverma}@us.ibm.com; A. Russo, Imperial College, London, UK; email: a.russo@imperial.ac.uk; C. Williams, The Defence Science and Technology Laboratory, Porton Down, UK; email: cwilliams@dstl.gov.uk.

0360-0300/2019/02-ART121 $15.00
https://doi.org/10.1145/3295749

devices. Controls need to be in place to manage these interactions so that they are not harmful to any of the individual or collective devices. Devices, especially mobile devices, will move into physically unprotected spaces and will have to interact with "unknown" devices [48]. However, in order for autonomous devices to effectively and efficiently carry out their tasks, proper policy-based management is critical. At a higher level, policies can be defined as directives given by a managing party to one or more managed parties in order to guide their behavior. Policies can be of different types, for example:

- Constraint policies govern actions executed by the managed parties, by which different actions are deemed allowed, not allowed, or obligatory. Access control policies [49] represent a well-known example of constraint policies.
- Goal-based policies state goals that the managed parties must achieve, for example, maintain a minimum threshold of utilization or try to finish a task before a specific deadline.
- Utility-based policies aim at producing the best outcome according to some value function, such as minimizing energy consumption.

However, because policies represent the key input for policy-based management, it is critical that they be "correct and fit for their use." This requirement has thus motivated research on methods and tools for policy analysis to determine whether sets of policies verify some given properties (e.g., consistency), and for assessing the impact of policy changes. We believe that in the context of present and future operations for autonomous devices, ensuring that policies are correct and adequate to address a large variety of situations is a critical requirement. Policy analysis is also crucial to support policy evolution as the analysis results can provide indications on how to best modify policies to fit continuously changing contexts and situations. In this respect, methods and tools to analyze policies and to support policy evolution and adaptation will be increasingly relevant. The goal of this survey is thus to provide a comprehensive view of existing methods and tools developed for policy analysis in different domains and for different analysis goals. Moreover, given that there are no policy analysis tools for autonomous devices in the IoT context and in other contexts, this survey aims at outlining, as conclusions, the key requirements for next-generation policy-based management systems. We emphasize that, as policies will be increasingly crucial for autonomous systems, analysis methods for policies will be critical.

The article is organized as follows. Section 2 briefly discusses the most relevant policy domains: access control and network management. Section 3 covers the properties that the different analysis methods address; Section 4 presents a comprehensive taxonomy of the analysis methods. Section 5 presents an overview of several well-known approaches and frameworks, while Section 6 compares them under different dimensions, including the analysis methods used. Section 7 outlines a few novel research directions.

2 POLICY DOMAINS

In this section, we briefly discuss policies in the two major areas in which policy-based management has been widely deployed: access control and computer network management.

2.1 Access Control Policies

Access control policies specify which subject (e.g., user, process, and application) can access which resources (e.g., files) for performing which actions (e.g., read, write). Many access control models have been proposed, including models that take time and location into account [47, 72] and models for privacy-sensitive data [183]. Access control mechanisms are embedded in many different systems, ranging from operating systems to database management systems. In what follows, we briefly describe a few relevant models and refer the reader to Bertino et al. [49] for details.

2.1.1 Role-Based Access Control (RBAC). The RBAC model consists of four components [84, 196]: users, roles, permissions, and sessions. A role represents an organizational function within a given domain, such as a coalition. Roles are granted permissions required for the execution of their functions. A permission consists of the specification of a protected object and an action,

The RBAC model definition includes several functions. The user assignment (UA) function specifies which user is assigned to which roles, whereas the permission assignment (PA) function specifies the set of permissions assigned to a role. In a particular system, when a user $u_i \in U$ becomes active, the *user* function assigns a session $s_i \in S$ to u_i, while the *roles* function maps s_i to the subset of roles that are associated with u_i. The following definition (adapted from Sandhu et al. [196]) formally defines the RBAC model.

Definition 1 (RBAC Model [47]). The model consists of the following components.

- U, R, P, S refer to the set of users, roles, permissions, and sessions, respectively.
- PA is the permission assignment function that assigns permissions to roles (i.e., $PA \subseteq R \times P$ and $PA(r_i) \subseteq P, \forall r_i \in R$).
- UA is the user assignment function that assigns users to roles (i.e., $UA \subseteq U \times R$ and $UA(u_i) \subseteq R, \forall u_i \in U$).
- The *user* function assigns a session to a single user (i.e., $user : S \longrightarrow U \mid user(s_i) \in U$).
- The *roles* function assigns a session to the roles associated with the user that activated the corresponding session (i.e., $roles \subseteq S \times 2^R \mid roles(s_i) = \{r \mid (user(s_i), r) \subseteq UA\}$).
- RH is the role hierarchy function (i.e., $RH \subseteq R \times R$), which refers to the partially ordered role hierarchy (written $\geq$).

The original definition of RBAC does not include the notion of "signed authorization" (i.e., positive or negative authorization). By default, all permissions granted to a role are positive authorization, but negative authorizations are useful when dealing with large sets of protected objects organized according to hierarchies. Signed authorizations have been widely investigated [191] and also introduced in access control systems of commercial products (e.g., the access control model of the SQL Server provides the negative authorization by the DENY authorization command SQL [9]). See [45] for a definition of RBAC with signed authorizations.

Example: *In a healthcare organization, there are different roles, including doctor, nurse, manager, and intern. As an example, the doctor role is responsible for predefined actions such as performing diagnosis, prescribing medication, ordering laboratory tests, and writing reports. Hence, we have* $Roles = \{doctor, nurse, manager, intern\}$, $Action_{doctor} = \{perform, prescribe, order, write\}$, *and* $Object_{doctor} = \{diagnosis, medication, laboratory\ test, report\}$. An example of a signed positive authorization policy is:

$$R :=< role : doctor, action : prescribe, object : medication, sign : + >.$$

RBAC has been extended in different directions. For example, Bertino et al. [47] introduced Temporal-RBAC (TRBAC) to address the challenge of the temporal dependencies among roles. Due to the tremendous increase in location-based services and mobile applications, the GEO-RBAC model was introduced that extends RBAC with geospatial information [72]. In GEO-RBAC, roles are activated based on the geo-location of users. For supporting both spatial and temporal constraints, Kumar and Newman [128] introduced the Spatial-Temporal RBAC model (STRBAC).

2.1.2 Extensible Access Control Markup Language (XACML). XACML [3] is an XML-based language by the OASIS standards organization for the specification of access control policies. In XACML, a policy is organized according to four elements: *subject*, *resource*, *action*, and *environment*. The *subject* is the entity requesting access with a particular type of *action* on a *resource* (e.g., data, service, or system component) within a context (i.e., *environment*); the policy either decides

to allow the request or to deny it. The XACML model can be characterized as an attribute-based access control (ABAC) model since the policy elements are associated with attributes (i.e., properties) that are essential for characterizing subjects and objects and making appropriate decisions based on these attributes. The main concepts of XACML policies are as follows.

- The *Policy Set* is a set of XACML policies while a *Policy* comprises a *Target* and a set of *Rules*.
- The *Policy Target* specifies the requests that are controlled by a particular policy. The *Target* specifies the attributes of the subjects, resources, actions, and environments that characterize the access requests.
- Each *Rule* is mainly composed of two elements; *Condition* and *Effect*. The Condition element specifies constraints on the request's attributes (i.e., subject, resource, action, and environment) while the *Effect* element specifies whether the request is allowed (*Permit*) or denied (*Deny*) based on the condition. A rule contains optionally the *Target* element, which also specifies the requests applicable to the corresponding rule. The result of enforcing a rule on an access request can take one of four possible values: Permit, Deny, Indeterminate (i.e., the decision cannot be made due to an error or some missing value), or Not Applicable (i.e., the request cannot be answered by this service).
- The *Rule Combining Algorithm* resolves the case when multiple rules are applicable to a request, but these rules have different effects on the corresponding request. XACML supports several conflict resolution strategies: permit-overrides (i.e., permit the request), deny-overrides (i.e., deny the request), deny-unless-permit (i.e., deny the request unless one of the rules permits it), permit-unless-deny (i.e., permit the request unless one of the rules denies it), first-applicable (i.e., apply the first rule in the policy file) and only-one-applicable (i.e., no decision is provided to the request when multiple rules are applicable).
- The *Obligation* is a function that can be executed before or after the policy is enforced on an access request. An example of such an obligation function is to log information about the corresponding policy and request.

Example: *A user can create a financial transaction if the user's credit balance is not lower than the amount of transaction and banking cost. Also, the transaction is valid during the office hours. This policy is modeled in XACML as follows:*

$< Policy\ PolicyId = \text{“}p1\text{”}\ RuleCombinationAlgId = \text{“}Deny - Overrides\text{”} >$
$< Rule\ RuleId = \text{“}r1\text{”}\ Effect = \text{“}Permit\text{”} >$
$< Target >$
$< subject > Bank < /subject >$
$< resource > Transaction < /resource >$
$< action > Create < /action >$
$< condition > transaction\_value + banking\_cost < credit\_balance < /condition >$
$< /Target >$
$< /Rule >$
$< Rule\ RuleId = \text{“}r2\text{”}\ Effect = \text{“}Permit\text{”} >$
$< Target >$
$< subject > Bank < /subject >$
$< resource > Transaction < /resource >$
$< action > Create < /action >$
$< condition > current\_day \in [Mon, Fri] \wedge current\_time \in [08AM, 6PM] < /condition >$
$< /Target >$
$< /Rule >$
$< /policy >$

We assume that the values of transaction_value, banking_cost, credit_balance, current_day, and current_time are constrained. The type of policy is Deny − Overrides, which returns permit if all rule evaluations return permit.

2.2 Network Policies

Network policies are sets of rules composed of conditions, constraints, and settings governing the operations (e.g., access to resources) within a network or across networks. Such rules allow one to specify which subject is authorized to connect to the network and the circumstances under which the subject can or cannot connect. Here, we discuss two types of network policies: firewall and software-defined networking (SDN). We discuss those two types separately for two reasons: (a) SDN policies are more general than firewall policies and (b) firewall policies have been the focus of a large body of research. Hence, in order to discuss firewall policies with adequate depth, we separate their discussion from SDN.

2.2.1 Firewall Policies. The firewall is a network element that filters traffic between network segments [58]. In particular, it filters out packets based on their characteristics and performs actions on the packets that do not match the firewall rules. The firewall thus contains a list of firewall rules (a.k.a. firewall policies) that specify the matching conditions for the traffic and the actions to be taken against the packets. Firewall rules follow the event-condition-action (ECA) paradigm, where an event (e.g., an incoming packet) triggers the automatic validation of the stated conditions and actions. Firewall rules are specified using network fields, including the protocol type, the IP address and port number of the source, and the IP address and port number of the destination. Commonly, a firewall rule follows the following format [21]:

$$Rule :=< order >< protocol >< src_{ip} > [< src_{port} >] < dst_{ip} > [< dst_{port} >] < action >,$$

where $order \in \mathbb{Z}^{\geq}$ (i.e., any non-negative integer), $protocol \in \{TCP, UDP, *\}$, src_{ip} and dst_{ip} express, respectively, source and destination IP address represented (in IPv4 format) as $address := x_1.x_2.x_3.x_4, x_i \in [0, \ldots, 255] \cup \{*\}$, $action \in \{allow, deny\}$, $port \in \mathbb{Z}^{\geq} \cup \{*\}$, and $*$ is used to denote the keyword *any* or a domain range.

If the traffic matches the specified filtering rule, the action of the rule is executed; otherwise, the next rule, as defined by the order sequence, is executed, and so on.

Example: *Consider the following firewall rules: "*$R_1 :=< 1 :$ *TCP*, 140.192.37.∗, *any*, ∗.∗.∗.∗, 80, *deny* $>$*" and "*$R_2 :=< 2 :$ *TCP*, ∗.∗.∗.∗, *any*, 161.120.33.40, 80, *accept* $>$ *."*

First, R_1 *(*$order = 1$*) denies all traffic from* src_{ip} *(140.192.37.*) and destined to any address on port 80. Then,* R_2 *(*$order = 2$*) accepts all traffic destined to* dst_{ip} *(161.120.33.40) on port 80. These firewall rules with this ordering have the effect of denying all traffic coming from* src_{ip} *(140.192.37.*) and destined to* dst_{ip} *(161.120.33.40) on port 80. However, the same traffic will be accepted if the order is reversed.*

2.2.2 SDN Policies. SDN is a networking model designed for existing network infrastructures to separate the control logic module from other underlying modules that control traffic forwarding (i.e., routers and switches) [167, 204]. The policy enforcement and network reconfiguration in an SDN architecture are thus significantly simplified. We refer the reader to Kreutz et al. [127] for a comprehensive survey and details of SDN.

OpenFlow [168] is a well-known technology that standardizes the communication between the switches and the software-based controller in an SDN architecture. OpenFlow specifies the flow table structure that describes how packets are processed in the switches. Specifically, a flow table is composed of a set of flow rules (i.e., policies) and each rule consists of three parts:

- A matching rule that comprises different fields, including switch port, source IP address, MAC address and port number; destination IP address, MAC address and port number; and VLAN tag
- An action that can be either dropping, forwarding, or modifying the matched packets
- Counters that record statistical data about the matched packets

The matching process starts with the arrival of a new packet; then, flow tables are searched sequentially for a matching rule. OpenFlow enables implementing the SDN policies in an assembly-like machine language. A number of benchmark tools (such as Cbench [2], OFCBenchmark [113], PktBlaster [10]) have been proposed to evaluate the performance of OpenFlow. Also, in order to enhance flexibility, several high-level networking languages have been suggested, such as FlowLog [180], Pyretic [174], Frenetic [86], NetCore [173], FML [102], and HFT [83]. There are several open-source implementations of OpenFlow Controller as well as commercial versions [5].

To improve OpenFlow, several tools have been developed. FLOWGUARD [104] was designed to detect and resolve security policy violations in OpenFlow-based networks. This tool employs five resolution strategies: flow removing, tendency breaking, update rejecting, flow rejecting, and packet blocking. VANT-GUARD [205] is an extension to OpenFlow; it addresses security challenges in OpenFlow by increasing data plane intelligence. VANT-GUARD protects SDN applications from saturation attacks. It also enhances detection and responsiveness to threats. Other tools, such as VeriCon [33], can verify the correctness of SDN programs on different topologies and for a sequence of network events.

Example: *Assume that the information about two subnetworks is as follows: Subnetwork* 1 (S_1): *100.0.1.0/24 and Subnetwork* 3 (S_3): *100.0.2.0/24. Mary is a developer* (*in* S_1) *allowed to access the resources at* S_3 *and the destination TCP port is Telnet. The policy can be written as*

$$Rule := p : path \mid (S_1\_S_3) \ \wedge \ protocol = TCP \ \wedge \ port = 24 \ \wedge \ Mary : user \in Developer.$$

3 ANALYSIS GOALS

Policy analysis is typically carried out for two different purposes: verifying the satisfaction of a set of quality requirements, and designing and organizing a set of policies. In what follows, we describe in more detail these policy analysis goals.

3.1 Assessment of Policy Quality

The notion of policy quality was introduced by Bertino et al. [45, 46] as a set of basic policy requirements. Ensuring the quality of a set of policies can be stated as the problem of making sure that the set of policies be consistent, minimal, relevant, complete, and correct. They should also minimize the exceptions that may occur at runtime. In what follows, we describe each of these quality requirements.

3.1.1 Consistency. Consistency (*CON*) refers to making sure that the policy sets do not include policies that contradict each other. For example, in the case of access control, ensuring consistency means making sure that a policy set does not include a policy allowing an access request and a policy denying the same request. Inconsistent policies lead to conflicts at policy enforcement. There are different types of conflicts [34, 70, 170], including modality conflicts (an inconsistency in policy specification of two policies that are applied to a request with different signs), conflict of duty (when a system fails to assert the separation of duty principle), conflict of interest, and conflicts of priorities.

3.1.2 Minimality. Minimality (*MIN*) refers to making sure that the sets of policies do not include redundant policies. For example, consider a file directory and an access control policy stating

that a user can read all the files in the directory. Then, assigning the same user a read permission on each single file in the directory would result in a policy set with redundant policies. Redundant policies increase the administrative work required to manage policies. For example, if a user is no longer required to access a given object, all policies controlling such an access must be properly modified, thus increasing the security risk if these changes are not properly propagated.

3.1.3 Relevance. Relevance (*REL*) requires that the sets of policies do not contain policies that do not apply to any action executed by the users. In the case of access control, irrelevant policies may undermine security. For example, an attacker may try to compromise a user in order to exploit the permissions of this user. Hence, making sure that users do not have permissions for accesses that they are not expected to execute minimizes the consequences of such exploitations.

3.1.4 Completeness. Completeness (*COM*) refers to making sure that all actions executed in the domain controlled by a policy-based management system are covered by some policies. For example, in the case of access control policies, ensuring that a set of policies is complete means ensuring that for each access request issued by a subject, there is a policy either allowing the access or denying it. Incomplete policy sets may lead to an unpredictable outcome and increase the cost of manual security administration.

3.1.5 Correctness. Correctness (*COR*) requires that the policies be free of faults and compliant with their intended goals and system requirements. Ensuring the correctness of policies includes validating their syntax and verifying their ability to achieve their goals in all possible contexts and scenarios.

3.2 Policy Design and Organization

In many cases, it is critical to maintain an optimal design and organization of policy sets in case of policy evolution or integration of policies from different organizations. To properly reorganize and evolve policy sets, it is often important to assess the similarity of different policies to determine whether one can consolidate similar policies into a single policy. It is also critical to assess the impact of policy modifications.

3.2.1 Policy Set Structuring. The policy set structure (*PSS*) has a primary effect on the cost complexity of policy management and policy enforcement. In particular, organizing the policies and rules in policy sets [156] helps in optimizing policy enforcement. Moreover, ensuring optimal structuring for policy components, which covers all potential components, affects the construction of policy management systems (e.g., role mining in RBAC systems [172]).

3.2.2 Similarity Analysis. The policy similarity analysis (*SA*) is the characterization of the relationships among a set of policies [131]. SA is an important step for policy integration and organizations collaboration since the collaboration can be planned based on the SA results of the collaborators' policies [165]. Several approaches have been proposed for policy SA including Mazzoleni et al. [164], Lin et al. [132], Mazzoleni et al. [165], and Lin et al. [131]. In particular, Mazzoleni et al. [165] defined a model for policy similarity based on the reaction of policies to a set of requests, while Lin et al. [132] and Lin et al. [131] proposed other models based on the policy structure and components.

3.2.3 Change Impact Analysis. Change impact analysis (*CIA*) refers to assessing and evaluating the extent of the change on the specifications of a set of policies by identifying the potential consequences and estimating the risks associated with the change. Thus, it provides an accurate understanding of the implications of a proposed change. *CIA* techniques typically evaluate the

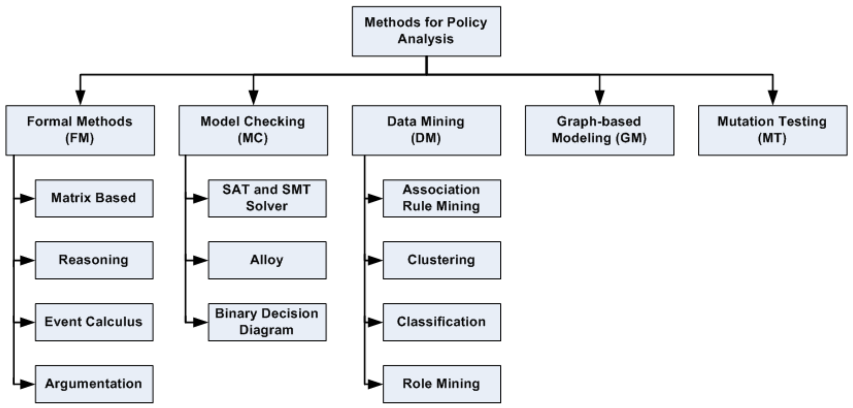


Fig. 1. Taxonomy of the methods used in policy analysis.

changes among two versions of a policy by providing a set of counterexamples that show semantic differences between the two policies.

4 TAXONOMY OF ANALYSIS METHODS

Various methods have been proposed for policy analysis, which we review in what follows. Figure 1 shows a taxonomy of these methods.

4.1 Formal Methods (FM)

A large body of research exists in the area of formal methods for policy analysis based on a variety of formalisms, including event calculus and argumentation-based reasoning. In what follows, we briefly summarize the formal methods most commonly used for policy analysis.

4.1.1 Matrix-Based Methods. A matrix is a collection of elements organized in rows and columns. A matrix is denoted by its dimensions (i.e., a matrix A that is composed of m rows and n columns is denoted as an $m \times n$ matrix), and an element in A is denoted as $a_{i,j}$.

In the context of policy analysis, matrices are used as a representation mechanism for the policy components. In particular, Boolean matrices (a.k.a. binary matrices or logical matrices) are mainly used to relate two finite sets according to 0 or 1 values. The existence of a relationship between two corresponding elements in the set is denoted by 1 and by 0 otherwise. For example, in RBAC [217], the user-role assignment is modeled by matrix A, whose dimensions are $m \times k$ and the role-permission association is modeled by matrix B, whose dimensions are $k \times n$. The Boolean multiplication of A and B (see Definition 2) returns the full set of user permissions.

Definition 2 (Boolean Matrix Multiplication [217]). A Boolean matrix multiplication between Boolean matrices $A \in \{0,1\}^{m \times k}$ and $B \in \{0,1\}^{k \times n}$ is $A \otimes B = C$, where C is in space $\{0,1\}^{m \times n}$ and

$$c_{ij} = \bigvee_{l=1}^{k} (a_{il} \wedge b_{lj}).$$

Moreover, the complete set of role hierarchies in RBAC can be represented using the transitive closure of roles [105]. For example, let R denote a set of roles and H denote the role hierarchy defined over R. Thus, the pair $(r_i, r_j) \in H$ implies that the role r_j is a child of the role r_i.

The transitive closure of H on R (see Definition 3) is the relation H^+ such that $(r_i, r_j) \in H^+$ means that there are several hierarchy levels between the roles r_i and r_j.

Table 1. Predicates Defined by the Specifications of Event Calculus [59]

Predicate	Description
$Happens(a, t)$	The event a occurs at a time point t.
$Initiates(a, f, t)$	The event a sets the fluent f to true at time t and after that.
$Terminates(a, f, t)$	The event a sets the fluent f to false at time t and after that.
$HoldsAt(f, t)$	The fluent f is true at time t.
$Clipped(t_1, f, t_2)$	Between the times t_1 and t_2, an event occurs that sets the fluent f to false.
$Declipped(t_1, f, t_2)$	Between the times t_1 and t_2, an event occurs that sets the fluent f to true.

Definition 3 (Transitive Closure). The transitive closure is a union set defined by the formula:

$$H^+ = \bigcup_{i \in \{1,2,3,\dots\}} H^i,$$

where H^i is the i^{th} power of H, defined inductively by $H^1 = H$, and for $i > 0$, $H^{i+1} = H \otimes H^i$.

Several algorithms for policy analysis have been proposed that use different forms of matrices (e.g., conflict matrix [223], and access control matrix [100]) while other analysis frameworks use matrices for policy representation in conjunction with other analysis techniques [217].

4.1.2 Reasoning Methods. There are various forms of logical reasoning, including deductive, inductive, and abductive reasoning. Deductive reasoning is the process of using one or more premises about the system behavior together with the history of events occurring in the system to reach a certain logical conclusion about the state of the system properties. Inductive reasoning can be defined as the derivation of a general conclusion about the system behavior based on premises on the observed history of events and the state of system properties. Abductive reasoning is a form of logical inference that starts with an observation of the system behavior to provide the best explanation using the sequence of events to reach a state of system properties. The conclusion of both inductive and abductive reasoning might be uncertain based on the premises given, whereas the truth of the deductive reasoning's conclusion is definite.

In the context of policy analysis, some approaches [215] use reasoning associated with an SMT solver (see Section 4.2.1). While other approaches [35, 38, 59] use reasoning utilizing the policy representation by Event Calculus (see Section 4.1.3). Another reasoning-based approach to analyze policies was proposed by Halpern and Weissman [96] based on a first-order logic approach that helps in reasoning about policies and preserving traceability.

4.1.3 Event Calculus. Event calculus (EC) is a logic-based formal model for expressing and reasoning about the occurrence of events. Being able to express the dynamic aspects of events, EC has been used for representing the specification of policies and analyzing them (e.g., [35, 38, 59]). EC has many variants, but the one widely used for policy analysis is the variant proposed by Russo et al. [194]. The specifications of this variant comprise a set of time-varying properties referred to as $fluents$, a set of positive integers referred to as $timepoints$, and a set of actions referred to as $events$. This variant also includes predicates describing the order of action, such as $Happens$, $Initiates$, $Terminates$, and $HoldsAt$, and some auxiliary predicates such as $Clipped$ and $Declipped$ (see Table 1).

4.1.4 Argumentation Methods. Argumentation is a method for reaching conclusions through logical reasoning on conflicting information [219]. In the artificial intelligence area, Dung [79] proposed an abstraction argumentation framework, later extended to logic-based [50] and

value-based [42] frameworks. In Dung's framework, argumentation is formally defined as a pair $<T, A>$, where T is a set of abstract arguments that represent data or a proposition and A is a binary attaching relation on the subsets of T where the relation shows the conflicts between the arguments. In particular, for any two arguments $t_1, t_2 \in T$, we say that t_1 attacks t_2 when the relation $(t_1, t_2) \in A$. Argumentation reasoning is given through the notion of an *admissible argument* as defined in Definition 4. In the framework $<T, A>$, T is represented in a logic language L.

Definition 4 (Admissibility of Arguments [37]). Given an argumentation framework $<T, A>$, an argument $t_i \in T$ is admissible $\Leftrightarrow (t_i, t_i) \notin A$ (i.e., t_i does not attack itself) $\wedge (\forall t_j \mid t_j \in T \wedge (t_j, t_i) \in A \wedge i \neq j \rightarrow (t_i, t_j) \in A)$ (i.e., for any other argument that attacks t_i, t_i counterattacks it).

In the context of policy analysis, argumentation policies can be modeled as arguments. The attack relation defined over the set of arguments serves as a criterion to deal with possible conflicts among policies and to select a set of compatible policies that achieve a set of requirements. Argumentation has been used for analyzing firewall policies [29, 36, 37] and access control policies [51, 52, 185].

4.2 Model-Checking (MC) Methods

The model-checking technique has been introduced to exhaustively and automatically verify the correctness of properties and specifications of concurrent finite-state systems. To achieve this goal, a mathematical representation is used to characterize the system's model and specifications. A typical model-checking system can be described by two components: (a) the preprocessor, which extracts a set of states S from a program and models them using a state transition graph M, and (b) the engine, which takes the state transition graph M and a temporal formula f and determines whether the formula is true or false (i.e., counterexample).

Model checking has some advantages compared with other verification techniques, such as automated theorem proving and proof checking [65]. However, the size of state transition graphs can be exponential, which is the main drawback of model checking. Therefore, several techniques—such as symbolic model checking, bounded model checking (e.g., SAT), compositional reasoning, abstraction, and partial order reduction—have been proposed to address the issue [82]. In those methods, the model-checking problem reduces to a graph search problem, and binary decision diagrams (BDD) or other temporal logic methods can be used for the traversal of the state space. Model checking has been widely used for policy analysis to detect, for example, conflicts between access control policies or security errors in access control policies.

In what follows, we describe model-checking methods that have been applied to policy analysis.

4.2.1 SAT and SMT Solvers. The Boolean satisfiability (SAT) problem is the problem of evaluating the satisfiability of a given Boolean formula. The SAT problem is formulated in Definition 5.

Definition 5 (The SAT problem [68]). Given a Boolean formula ϕ composed of

- n Boolean variables: $x_1, x_2, \ldots, x_n$;
- m Boolean connectives: any Boolean function such as $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT), $\rightarrow$ (implication), $\Leftrightarrow$ (if and only if); and
- Parentheses: One pair of parentheses per a Boolean connective.

The Boolean satisfiability problem posed on ϕ aims at finding some values that can be assigned to the variables composing ϕ to make its value *true*. If such values exist, then ϕ is satisfiable. Otherwise, ϕ is unsatisfiable.

The SAT problem is known to be NP-hard [67]. However, there has been tremendous progress (see [230]) in developing efficient mechanisms for solving the SAT problem. These algorithms are known as SAT solvers. Some SAT solvers are complete (e.g., the Davis-Putnam-Logemann-Loveland (DPLL) algorithm [74, 75]), while others are stochastic (e.g., random walk-based algorithms, such as WALKSAT [200]). For any given instance of a SAT problem, a complete solver is able to find the variable values that solve the problem in case it is satisfiable or proves that it is unsatisfiable. Meanwhile, a stochastic SAT solver is not able to prove the unsatisfiability for some instances.

Advances in SAT solvers have made them attractive underlying reasoners for policy analysis w.r.t. propositional logic formulas used to model various access control policies. However, native SAT solvers do not support efficient reasoning over non-Boolean variables, such as temporal constraints, which play a significant role in analyzing the correctness of policies. These non-Boolean variables are often left uninterpreted, hence restricting analysis capabilities [215]. An extension of SAT is *Satisfiability Modulo Theories* (SMTs) [39] which extend the Boolean SAT representation with additional operators, such as linear arithmetic and equality. Consequently, various SMT solvers were proposed (e.g., Z3 [77] and openSMT [7]) that support a more efficient fine-grained analysis than native SAT-based policy analysis tools.

Since policies are essentially a set of constraints over a set of resources, the policy analysis can be intuitively mapped into logic formulas. As a consequence, satisfiability checking can be employed to validate requirements, such as security requirements, over a given set of policies. Several algorithms and frameworks for policy analysis have been developed that use SAT [106, 116, 131] or SMT [25, 26, 114, 215] solvers.

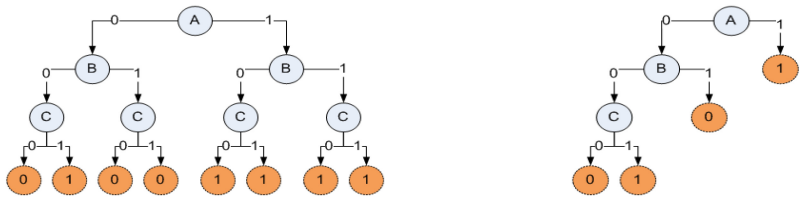
4.2.2 Alloy. Alloy is a declarative logical language for expressing structural constraints and behavior of a system [1, 111]. The language is simple and based on standard first-order logic. Its syntax uses the following notations:

- *Signatures*: A signature is a data type in Alloy. It can be considered equivalent to a class in object-oriented languages since a signature can be instantiated.
- *Relations*: A relation is a tuple that maps instances of signatures to each other.
- *Functions*: A function maps one instance of a signature to an instance of another signature.

When modeling system constraints using the Alloy language, the Alloy Analyzer [112] is executed to find structures that satisfy the constraints. Thus, the Alloy Analyzer can be used both to explore the model by generating sample structures and to check the properties of the model by generating counterexamples. The core of the analyzer engine is reduced to a Boolean SAT solver.

Since policy constraints can be modeled using the Alloy language, several frameworks for policy analysis use Alloy as the underlying building block [99, 101, 120, 154, 187, 198].

4.2.3 Binary Decision Diagrams. A binary decision diagram (BDD) is a data structure for representing a Boolean function that takes Boolean arguments as inputs and returns a Boolean output [20]. A BDD is represented as a rooted, directed, acyclic graph $G(N, E)$ composed of nodes N and edges E. The set of nodes N are categorized into two types: decision nodes corresponding to the Boolean arguments and terminal nodes corresponding to the Boolean decision (0-terminal and 1-terminal). The edges represent the Boolean values branching from nodes. Consider the Boolean function f defined in Equation (1) [20]. Figure 2 shows two examples of BDD representing f. Given n arguments in a Boolean function, there are 2^n different possible inputs; thus, an instance of BDD contains at most 2^n possible paths corresponding to the truth table for f (e.g., Figure 2(a)). For efficiency, the BDD is designed with the minimum number of paths by omitting the redundant



(a) A Diagram Derived from the Logic Truth Table (b) A Simplified Diagram

Fig. 2. Two binary decision diagrams of Equation (1) [20].

test of Boolean arguments (e.g., Figure 2(b)).

$$f = A \vee \neg B \wedge C \tag{1}$$

The BDD structure has been extended to the multi-terminal BDD (MTBDD) structure [87] where a terminal node is allowed to have a finite set of values instead of binary values (i.e., 0 or 1). The MTBDD structure is also known as an algebraic decision diagram (ADD) [32]. The implementation of both BDD and MTBDD is provided by the CUDD package [208]. BDD and MTBDD are used as the underlying representation of policies that facilitate the analysis process (e.g., [56, 85, 131, 186, 228]).

4.3 Data Mining (DM) Methods

Data mining is a technique for exploring massive datasets and finding interesting trends or patterns to guide decisions for further analysis. DM algorithms can be classified as supervised and unsupervised. Supervised algorithms require a learning stage on a historical dataset for building a model that summarizes the identified patterns. Unsupervised algorithms detect patterns in datasets without the need of the learning phase. Examples of supervised algorithms include classification while unsupervised algorithms include association rule mining and clustering. Some of the DM methods are described in what follows.

4.3.1 Association Rule Mining (ARM). ARM [17] is a technique for detecting statistical relations—referenced as *association rules* defined in Definition 6 (adapted from Tan et al. [213])—between items in a database. The mechanism aims at discovering all rules that satisfy a certain threshold on two metrics: *support* and *confidence*. The support metric indicates how often a rule applies to a given dataset; hence, it is used to eliminate uninteresting rules and capture the frequency of the rule. The confidence metric represents the quality of the rule. Hence, it measures both the reliability of the inference made by the rule and the strength of the relation between the items sets. The inference indicated by the association rule suggests a strong co-occurrence relationship between items in the antecedent and consequent of the rule. There are various algorithms designed for ARM, including the Apriori Algorithm [18], FP-growth [97], and Eclat [229].

Definition 6 (Association Rule [213]). Given a set of items $I = \{i_1, i_2, \ldots, i_n\}$ and a database D composed of a set of transactions T where each transaction $t \in T$ is composed of a set of items $t \subseteq I$, an association rule is an implication expression of the form $X \longrightarrow Y$ where X and Y are disjoint subsets of I (i.e., $X \subseteq I$, $Y \subseteq I$, and $X \cap Y = \phi$). The association rule $X \longrightarrow Y$ has two metrics:

- The support value s, which denotes the fraction of transactions containing both X and Y in the whole database D (i.e., $s(X \longrightarrow Y) = p(X \cup Y)$); and
- The confidence value c, which denotes the fraction of transactions that contain both X and Y among the transactions that contain X (i.e., $c(X \longrightarrow Y) = p(X \cup Y)/p(X)$).

Then, in an association rule ($X \longrightarrow Y$), where X and Y are attribute values, a confidence of the rule is conditional probability of X given Y, $Pr(X|Y)$ and support is the prior probability of X and Y, $Pr(X\ and\ Y)$. The algorithm used minimum support and minimum confidence to identify the largest itemset and the best rules.

The ARM-based policy analysis techniques explore large datasets to identify interesting rules for further analysis. For example, Ma et al. [153] used ARM in the context of policy analysis to classify constraints in RBAC. Cardinality constraints on the number of users, roles, and permissions are represented in terms of association rules. Bauer et al. [41] employed ARM to anticipate misconfiguration in access control policies. By applying ARM to the history of accesses, association rules are identified (the rules are mined by the Apriori algorithm [18]). Based on the rules, the data is analyzed and potential misconfigurations of access control policies are detected. Golnabi et al. [89] proposed an ARM-based technique to detect firewall anomalies defined in terms of the superset, subset, and correlation relationship among criteria. Firewall log files are mined to extract attributes (such as protocol, direction, source IP, destination IP, source port, and destination port).

4.3.2 Clustering. Clustering is a mechanism for distributing a set of objects into groups (referred to as *clusters*) where the intra-similarity among the objects of a group is higher than the inter-similarity with the other groups. The clustering technique defines the groups of objects based on some measure of inherent similarity or distance (e.g., Euclidean and Manhattan distance). Each cluster has a centroid object that identifies the corresponding cluster. After constructing the clusters, a new object is assigned to the cluster whose centroid is the closest one.

Well-known clustering algorithms include k-means [149], mean-shift [66], and spectral clustering [181]. In addition to the basic clustering techniques, hierarchical clustering builds a hierarchy of clusters. Hierarchical clustering follows two strategies: bottom-up approach (referenced as *agglomerative* [207]), and top-down approach (*divisive* [121]). Several open-source libraries (e.g., scikit-learn [184] and Weka [95]) provide implementations in different languages of these algorithms.

Clustering-based approaches partition access control policies into clusters. For example, for XACML policies in distributed applications, after extracting the rules in each policy, similar rules (identified based on some similarity scores) are assigned to the same cluster [19, 44]. Each cluster is separately analyzed for quality (such as redundancy, and inconsistency). A k-means clustering method [222] has been proposed by Marouf et al. [155, 156] to optimize the evaluation of XACML policies in the case of large numbers of requests. This approach clusters similar subjects (i.e., subjects in one cluster share a large number of policies relevant to all of them) to find relevant policies for incoming requests with the best rule orders. The best ordering is dynamically identified based on incoming requests, history of requests, and executions. Moreover, Benkaouz et al. [44] proposed a K-nearest neighbors-based technique for RBAC policies to enhance flexibility and reduce policy dimensionality in large-scale applications.

4.3.3 Data Classification. Classification is a learning technique for assigning an item to a set of predefined classes or groups (a.k.a. labels). Classification algorithms include two stages: training and prediction. In the training stage, the algorithm discovers the patterns of the attributes characterizing the items for each predefined class and summarizes these patterns in a training model. In the prediction stage, the algorithm uses the training model to predict the class of a new item.

Data classification techniques can handle complicated Boolean expressions, missing and continuous attributes. Unlike the formal logic approaches for inconsistency detection, classification techniques do not suffer from exponential growth and computational complexity [203]. In data classification, each rule (R_i) is represented by a set of attributes ($A_1, A_2, \ldots, A_n$) based on

which decisions are made (e.g., role, subject) and a class of the rule (C) (e.g., allowed, denied), ($R_i : A_1 \wedge A_2 \wedge \ldots \wedge A_n \longrightarrow C$). Data classification techniques—such as C4.5 [190], ID3 [189], limited search induction algorithm (LSIA) [55], and ASSISTANT'86 [57] (with some modifications)—are mainly used for detecting incompleteness [201, 202] and inconsistency [31, 201, 203], which are formally defined in Definitions 7 and 8. In Definition 7, incompleteness is defined for attribute-based policies. Thus, a complete policy set requires the rules in the set to cover all possible values that the attribute domains can take. However, such a requirement may be difficult to meet, especially when the domains are infinite or the domains are not fully known in advance. Hence, approaches that analyze the actual system behavior would be more suitable.

Definition 7 (Incompleteness [201]). Given a rule set $\mathfrak{R}$, a rule $R_i \in \mathfrak{R}$, and a set of attributes A, let $\Upsilon(A_k)$, $A_k \in A$ denote the set of all possible values that can be assigned to attribute A_k and let $v(R_i.A_k)$ denote the set of values assigned to attribute A_k in rule R_i. $\mathfrak{R}$ is incomplete with respect to A_k if and only if:

$$\bigcup_{R_i \in \mathfrak{R}} v(R_i.A_k) \subset \Upsilon(A_k).$$

Definition 8 (Mutual Inconsistency [201]). Given a rule set $\mathfrak{R}$ and two rules $R_i, R_j \in \mathfrak{R}$, let $v(R_i.A_k)$ and $v(R_j.A_k)$ denote the set of values assigned to attribute A_k in rules R_i and R_j, respectively. R_i and R_j are mutually inconsistent if and only if:

$$1.\ \forall A_k \in A,\ v(R_i.A_k) \cap v(R_j.A_k) \neq \phi \ \textit{and}$$

$$2.\ v(R_i.Permission) \neq v(R_j.Permission).$$

4.3.4 Role Mining. Role mining (RM) is an application of DM specifically designed for transforming a non-RBAC system into an RBAC system by finding user roles in the observed policies [217]. The RM problem is formally defined in Definitions 9 and 10. RM is a tool for examining whether roles are appropriately assigned to existing functions and business processes.

Definition 9 (Role Mining Problem [217]). Given a set of users U, a set of permissions P, and a user-permission assignment UPA, find the minimal set of roles R where user-to-role assignment UA, a role-to-permission assignment PA, and user-to-permission assignment UPA are 0-consistent (i.e., $\delta = 0$).

Definition 10 (δ-Consistency [217]). A given user-to-role assignment UA, role-to-permission assignment PA and user-to-permission assignment UPA are δ-consistent if and only if:

$$\| M(UA) \otimes M(PA) - M(UPA) \|_1 \leq \delta,$$

where $M(UA)$, $M(PA)$, and $M(UPA)$ denote the matrix representation of UA, PA, and UPA, respectively. δ-consistency bounds the degree of difference between the UA, PA, and UP.

RM is an NP-hard problem [217] for which various algorithms have been proposed. In particular, Schlegelmilch and Steffens [199] designed the ORCA RM tool, which is based on a hierarchical clustering on permissions. Another algorithm is RoleMiner (proposed by Vaidya et al. [218]), which is based on subset enumeration. Moreover, Molloy et al. [172] designed an RM algorithm that considers the semantic meanings of the roles. In addition to being a method for policy analytics, RM can be used to migrate from or to an RBAC model [212, 227].

4.4 Graph-Based Modeling (GM) Methods

In the context of policy analysis, a set of policies can be represented as a graph (i.e., nodes connected with edges) or a tree (a special form of a graph), which helps to efficiently query, analyze, and verify the corresponding policies by using the graph operations. In particular, graph traversal, graph union, graph intersection, and graph difference are used to analyze properties of policies. The preparatory step of the analysis process is to transform a given policy set into a single graph (referenced as *Policy Graph*) or tree (referenced as *Policy Tree*). Transforming the set of policies into a graph includes modeling the policy components (e.g., roles in RBAC policies) into nodes and constructing a path (i.e., a sequence of edges connecting certain nodes) for every policy. The transformation includes multiple graph operations, such as traversing and updating the graph. The graph-based policy model has been used for ensuring that the specifications of the policies comply with their goals and the system requirements [14, 21–24, 27, 28, 45, 76, 210, 226].

4.5 Mutation Testing (MT) Methods

Mutation testing [78] is a technique for software testing that involves generating several versions of the software by making one small change at a time. Mutation testing uses mutation operators that differ based on the corresponding programming language. Each generated version is called a mutant; the goal is to create test cases that are able to detect all faults in the mutants. A major limitation of mutation testing is its computational cost. Hence, several algorithms have been proposed for reducing the number of mutants without affecting the effectiveness of tests. This problem is referred to as the *Mutant Reduction Problem* [118] as described in Definition 11. Several techniques were introduced to address this problem, including mutant sampling [15, 54], mutant clustering [107], selective mutation [163], and higher-order mutation [117].

Definition 11 (The Mutant Reduction Problem [118]). For a given set of mutants M, and a set of tests T, $MS_T(M)$ denotes the mutation score of the test set T applied to mutants M. Find a subset of mutants M' from M, where $MS_T(M) \approx MS_T(M')$.

Mutation testing has been adopted for policy testing and verification [159–162, 177, 178, 188] in three consequent steps: (1) inject faults into the original policies to generate a set of mutant policies; (2) create various scenarios to enforce both original and mutant policies; and (3) measure the percentage of mutant policies whose enforcement output is different from their original policies. The percentage calculated in the last step is referred to as *detection percentage of mutant policies*, which indicates the correctness level of the policy set. Consequently, mutation testing helps in discovering certain scenarios where the corresponding policies are specified incorrectly. To generate mutant policies for mutation testing, it is crucial to design an efficient mutation mechanism (known as *mutation operators*). Several researchers [81, 130, 175, 176] defined generic operators that are independent of the policy model.

5 SYSTEMS AND FRAMEWORKS FOR POLICY ANALYSIS

In this section, we survey several well-known systems and frameworks that address the challenge of policy analysis in various domains.

5.1 Environment for XACML-Based Policy Analysis and Management (EXAM)

EXAM is a comprehensive framework for policy analysis composed of five components: policy annotator, policy filter, policy similarity analyzer, and policy integration framework. The policy annotator preprocesses policies by adding annotations to represent the semantics of the policies [131]. The policy filter evaluates the similarity of each pair of policies using a policy similarity measure [132]. The similarity measure for a pair of policies produces a similarity score that is used

for reducing the number of analyzed policies by eliminating the low-score policy pairs. The policy similarity analyzer supports three types of analysis queries: (a) policy metadata queries related to the metadata associated with the policies, (b) policy content queries related to the content of policies, and (c) policy effect queries related to the interactions among the policies and their outcomes. The analyzer component is based on two methods: MTBDD and SAT solver. The policy integration represents policies in the MTBDD model to combine policies based on the use of algebraic operations defined by the fine-grained integration algebra (FIA) [192].

5.2 Margrave

Margrave is a software suite for analyzing XACML policies [85]. It uses the MTBDD structure as the underlying representation of policies. Margrave has two components: verification and change impact analysis. The verification component is an engine for querying policies and evaluating a particular property. The change impact analysis component uses a decision diagram, referred to as a *change-analysis decision diagram*, to analyze the changes among a pair of policies and outline the semantic differences between them. The analysis results are represented by a MTBDD that enables exploring them by different verification queries.

5.3 MIsconfiguRAtion ManaGEr (MIRAGE)

MIRAGE is a management framework for the analysis of network policies deployed in network devices (e.g., firewalls, intrusion detection systems, and VPN routers) [88]. The framework detects anomalies and verifies consistency, relevancy, and correctness of network security policies by using bottom-up and top-down approaches. The bottom-up approach analyzes the configuration errors in deployed components and the top-down approach refines global security policies into configuration of security components. These approaches are mainly based on representing policies using a formal model and using Boolean functions and reasoning methods. MIRAGE is composed of four services: intra-component analysis, inter-component analysis, aggregation mechanism, and refinement mechanism. The intra-component analysis detects the inconsistencies between the policies of a single security appliance, while the inter-component analysis detects the inconsistencies between the policies of different security appliances. The aggregation mechanism merges the policies of all security appliances into one global minimal consistent set of policies. The refinement mechanism deploys the aggregated global set of policies to newly added security appliances.

5.4 Policy Conflict Analysis for Quality of Service Management

Charalambides et al. [59] proposed a policy management technique to handle conflicts in the network management domain with a focus on quality of service (QoS) management. It classifies conflicts into application-specific (e.g., routing conflict) and domain-independent (e.g., redundancy and mutual exclusion) conflicts. To detect conflicts, it uses abduction and explanation of conflict occurrence. The detection approach concentrates on policies for network dimensioning (ND) (which is a component of the TEQUILA framework). It defines conflict rules for the Ponder [73] language; it then searches the policy repository to check whether there is a policy violating conflict rules.

5.5 A Toolkit for Firewall Modeling and Analysis (FIREMAN)

FIREMAN [228] is a framework for firewall modeling and analysis. FIREMAN provides static analysis methods to detect misconfigurations, inconsistencies, redundancies, and irrelevancies of firewall rules by modeling them in BDDs. FIREMAN can be used for both individual and distributed firewall systems. Thus, it supports analysis services at various levels: intra-firewall, inter-firewall, and cross-path. The analysis framework adopted by FIREMAN is a bottom-up strategy by

employing a local analysis for each individual firewall and then performing a global analysis considering the interactions between the distributed set of firewalls.

5.6 FLCheck

FLCheck is a framework to specify, enforce, and verify access control policies [56]. FLCheck uses BDDs for performing the analysis of access control policies. In particular, it analyzes the RBAC properties (e.g., consistency, safety, and domain-dependent conditions) and indicates how domain-dependent assumptions can be merged into the check without a need to modify policies.

5.7 Event Calculus Framework for Policy Specification and Analysis

Bandara et al. [38] proposed an approach to formalizing policy specifications and system behavior rules using EC. Through the proposed formal representation and using abductive reasoning techniques, it performs policy analysis to generate a refined set of policies while ensuring correctness, consistency, minimality, and completeness of the generated policy set. The approach aims to achieve consistency by detecting different actual and potential types of conflicts: modality conflict, conflict of duty, conflict of interest, and conflicts of priorities.

5.8 Analysis Tools for SELinux Security Policy

Different policy analysis tools, such as SLAT [94] and PAL [197], have been developed to analyze the security policies of SELinux [150]. Security-Enhanced Linux Analysis Tools (SLAT) [94] uses formal methods and model-checking techniques to verify that SELinux policies achieve their intended security goals. It translates access control policies written in SELinux's policy language into logic programs and then uses query evaluations for analysis. Policy Analysis using Logic-Programming (PAL) [197] uses an information flow model of SLAT for policy analysis. Functionalities of SLAT and PAL are similar, though they differ with respect to query languages. To represent queries, SLAT uses a special-purpose language determining information flow paths between security contexts. PAL, unlike SLAT, is implemented in XSB (i.e., it is a logic-programming and deductive database system based on tabled resolutions) [8]. To improve the understandability of SELinux policy analysis results, Xu et al. [225] developed a visualization tool based on both adjacency metrics and semantic substrates (i.e., a visualization technique that uses the user-defined semantic substrate to generate graph layouts) [206]. The framework enables administrators to detect possible policy violations by running visualization-based queries on the policies.

5.9 Graph-Based Policy Analysis (GPA)

GPA [226] is a framework for analyzing information flow in networks, representing policy queries, and identifying integrity violation of SELinux policies. GPA displays policy layouts by using two visual techniques: semantic substrates and adjacency matrices. It allows system administrators to apply a specific query on policies. Then, a policy violation graph (generated by the framework) displays violations recognized by an integrity model. The integrity model is developed based on the trusted computing base (TCB), the notion of transaction procedure in the Clark-Wilson security model, and Biba concepts. GPA introduces filtering and ignoring techniques to remove violations from policy graphs.

5.10 Tierless Programming and Reasoning for Software-Defined Networks

Nelson et al. [180] proposed an SDN-based declarative programming language, called Flowlog. Flowlog enables the writing of SDN policies as well as their analysis. In particular, it allows converting a Flowlog ruleset to Alloy (see Section 4.2.2) specifications. Upon compiling Flowlog rules into Alloy, their correctness is verified using the Alloy analyzer.

5.11 Access Control Policy Evaluator and Generator (AcPeg)

AcPeg [232] is a Java-based framework that enables analyzing policies using model checking [231]. The algorithm uses the symbolic model-checking approach [169] to support various goals, including checking whether the policies provide enough permissions to users and identifying policy weaknesses. For analyzing access control policies with this framework, they have to be specified in a formal language, called RW (Read-Write) [92]. However, the framework also supports the translation of the description of a policy from the RW language into the XACML format.

5.12 MOHAWK

Jayaraman et al. [115] introduced a framework called MOHAWK that enables error detection in ARBAC [195] access control policies. The policy analysis algorithm supported by MOHAWK combines both abstraction-refinement and bounded model-checking techniques. The algorithm first applies abstraction to a set of input policies (with successive refinements, if necessary) to efficiently evaluate an abstract form of these policies. The level of abstraction can be configured based on the scale of the access control policies (e.g., for a small number of policies, the user can configure the tool to abstract less). After the abstraction step, the algorithm applies model checking for verifying the abstract version of policy against a set of security properties. For this purpose, MOHAWK translates the abstract form of the policy to NuSMV specifications [6]. NuSMV is a model-checking library that uses both types of model-checking techniques, BDD and SAT.

5.13 Firewall Policy Advisor (FPA)

FPA [21–24] is a Java-based tool for analyzing firewall policies using a tree-based method. FPA supports two approaches for analyzing policies: intra-firewall and inter-firewall. The intra-firewall analysis approach evaluates policies within a single firewall while the inter-firewall analysis assesses policies between inter-connected firewalls. The analysis approaches aim at detecting inconsistency, redundancy, and irrelevancy in the policy set. In addition, FPA includes an editor that supports managing firewall policy rules. It also displays the appropriate order of the added or modified rules and the effect of the removed rules on the policy set.

5.14 Structured Firewall Design

A firewall is often designed as a sequence of rules leading to three main problems: consistency, completeness, and compactness [90]. To address these problems, Gouda and Liu [90] proposed a method called *structured firewall design* based on a firewall design diagram (FDD) instead of a sequence of rules that often conflict with each other. An FDD is an acyclic and directed graph. Two algorithms, FDD reduction and FDD marking, are used to combine rules. A firewall compaction algorithm to remove the redundant rules has also been proposed [134, 146]. The FDD is used to represent the semantics of a firewall. The FDD reduction algorithm is used to reduce the number of decision paths (i.e., generated rules) in an FDD. In the proposed method, the user deals only with the firewall decision diagram, which is a formal specification of the firewall. This requires the user to know the formal specification of the firewall, making the method less generic. Moreover, the FDD reduction is based on a graph traversal method that may be time consuming if the FDD specified by the user is very large.

Owing to a large number of firewall rules in a firewall, analyzing and understanding the firewall rules is a difficult task. An effective way to analyze and understand deployed firewall rules is by issuing queries. An SQL-like query language called *Structured Firewall Query Language* (SFQL) has thus been proposed [143, 144]. Furthermore, an algorithm for efficiently processing firewall

queries has also been proposed by Liu and Gouda [143]. The FDD is used as the core data structure by the query processing algorithm and engine.

The correctness of firewall rules and configuration are crucial for an effective security perimeter deployment. However, ensuring the correctness of firewall policies is a challenging task, since a firewall may have a large number (e.g., hundreds to a few thousand) of rules and firewall rules are often added at different times by different firewall administrators for various reasons. To address this challenge, Hwang et al. [109] proposed a systematic structural testing approach. The proposed method is based on the firewall policy coverage concept, used to test a firewall policy's structural entities (i.e., rules, predicates, and clauses). The method allows one to check whether each entity is specified correctly. Three structural coverage measurements, which monitor whether rules, predicates, or clauses are covered when evaluating packets against the policy under test, have been defined [109]: (1) *rule coverage measurement* is the percentage of the number of covered rules (i.e., predicates being evaluated to true) in a policy; (2) *predicate coverage measurement* is the percentage of the number of covered predicates (i.e., predicates being evaluated to true or false); and (3) *clause coverage measurement* is the percentage of the number of covered true or false values of clauses. Instead of exhaustively testing all possibilities, these measurements enable the testing to cover only specific entities.

A configuration of firewall very often leads to errors (i.e., misconfiguration). Such misconfiguration, considered as a policy fault, either creates security holes that allow malicious traffic into private networks or blocks legitimate traffic and disrupts normal services. A faulty firewall policy misclassifies some packets, thus leading to unexpected decisions. In [63, 64], a fault model is proposed for firewall policies, including five types of faults: *wrong order*, *missing rules*, *wrong decisions*, *wrong predicates*, and *wrong extra rules*. For each type of fault, an automatic correction technique is proposed. An approach that employs these techniques is proposed to automatically correct all or part of the misclassified packets of a faulty firewall policy.

5.15 Gorgias-B

Gorgias-B [4, 209] is a tool that combines argumentation reasoning with preference-based rules and abductive logic programming for making informed decisions. It can help users not only to define their decision policy but also execute scenarios to test it. Gorgias-B has been used for conflict resolution in various applications such as data access control [4], autonomous systems (e.g., Drones), and firewall configuration management [37].

6 COMPARISON

In this section, we compare 133 research publications describing either a system, framework, or a research approach for policy analysis. Our comparison considers three dimensions (see Table 3): policy domain, analysis goals[1], and analysis methods. The abbreviations for both policy analysis goals and methods used in the comparison table (Table 3) are listed in Table 2.

Figure 3 shows some statistics about the set of surveyed articles. Among the surveyed articles, access control is the policy domain that is the focus of the majority of them. In particular, RBAC and XACML are the models that are widely considered. Articles related to the network policy domain consider mainly the firewall rules model. Thus, research is needed for methods and tools to analyze other network policy models, such as the rule models of SDN. With respect to the analysis goals, most systems and frameworks focus on consistency and correctness. Therefore, research is needed on methods for analyzing completeness and relevancy, which are critical in dynamic contexts.

[1]If the field related to the analysis goals (i.e., policy quality or policy design) is marked as "N/A," this means that the policy analysis approach proposed by the corresponding research work does not address that goal.

Table 2. Symbols for Both Policy Analysis Goals and Methods

Policy Analysis Goals		Policy Analysis Methods	
Symbol	**Description**	**Symbol**	**Description**
CON	Consistency	*FM*	Formal Methods
COM	Completeness	*MC*	Model Checking
MIN	Minimality	*DM*	Data Mining
REL	Relevance	*GM*	Graph-Based Modeling
COR	Correctness	*MT*	Mutation Testing
PSS	Policy Set Structuring		
SA	Similarity Analysis		
CIA	Change Impact Analysis		

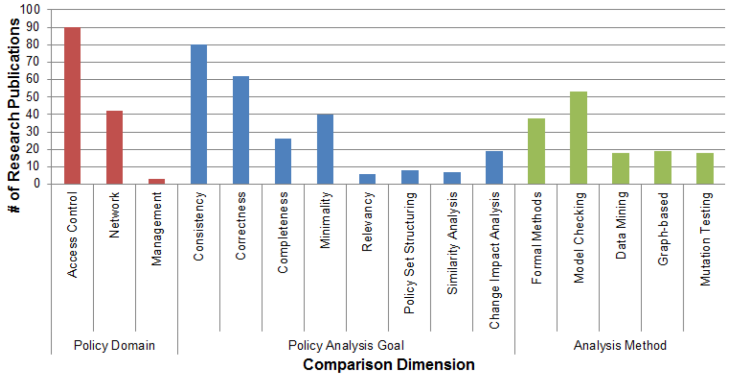


Fig. 3. Statistics about the 133 research publications describing the systems and frameworks compared in Table 3.

Also, analysis techniques addressing policy design and organization (e.g., similarity analysis and change impact analysis) need to be further explored. With respect to the analysis methods, the most used techniques are model checking and formal methods since these methods can be used for almost all of the analysis goals. DM is the least used technique because it is computing intensive, time-consuming, and error prone.

None of the policy analysis systems and frameworks is able to support all of the analysis goals. In particular, none of the systems is able to support the analysis of policies with respect to all policy quality requirements. In the access control policy domain, ProFact [14, 45][2] considers all quality requirements except correctness, while in the network policy domain, the Mirage [88] and FPA [21–24] frameworks consider all quality requirements except completeness. Figure 4 shows the association ratios of the analysis goals in the surveyed publications. All analysis goals are tightly associated with consistency, which is the focus of the majority of the articles, except for the goal of change-impact analysis that is tightly associated with correctness. This is owing to the similarity between the analysis goals of both correctness and change-impact analysis in terms of purpose and underlying algorithms. In addition, all goals are associated loosely with the similarity analysis

[2]ProFact is a policy analysis framework that uses provenance metadata in addition to the set of access control policies. Provenance metadata is a rich set of logged information about the system behavior at runtime [11–13].

Table 3. Comparison of Systems and Frameworks for Policy Analysis

Publications	Policy Domain	Policy Quality	Policy Design	Method
MIRAGE [88]	Network—Firewall	*CON*, *MIN*, *REL*, *COR*	N/A	FM—Reasoning
TEQUILA [59, 60]	Network	*CON*, *MIN*	N/A	FM—Event Calculus, FM—Reasoning
Wu et al. [223]	Access Control	*CON*, *MIN*	N/A	FM—Matrix based
Vaidya et al. [217, 218]	RBAC	N/A	*PSS*	FM—Matrix based, DM—Role Mining
Bandara et al. [38]	Access Control, Management	*CON*	N/A	FM—Event Calculus, FM—Reasoning
Bandara et al. [35]	Network	*CON*	N/A	FM—Event Calculus, FM—Reasoning
Bandara et al. [36, 37]	Network—Firewall	*CON*, *MIN*, *COR*	N/A	FM—Argumentation
Bandara et al. [34]	Access Control	*CON*, *COR*	N/A	FM—Reasoning
Turkmen et al. [215]	XACML	*CON*, *COR*	*CIA*	FM—Reasoning, MC—SMT Solver
Applebaum et al. [29]	Network—Firewall	*MIN*	N/A	FM—Argumentation
Boella et al. [51, 52]	Access Control	*CON*	N/A	FM—Argumentation
Moffett and Sloman [170]	Management	*CON*	N/A	FM—Reasoning
Kolovski et al. [126]	XACML	*CON*, *MIN*, *COR*	*SA*	FM—Reasoning
Lin et al. [132]	XACML	N/A	*SA*	GM—Tree, DM—Clustering
Lupu and Sloman [151]	RBAC	*CON*	N/A	FM—Reasoning
McDaniel and Prakash [166]	Access Control	*CON*, *COR*	N/A	FM—Reasoning
Craven et al. [69, 70]	XACML	*CON*, *COM*, *MIN*	*SA*	FM—Event Calculus, FM—Reasoning
Halpern and Weissman [96]	Access Control	*CON*	N/A	FM—Reasoning
Benferhat et al. [43]	RBAC	*CON*	N/A	FM—Reasoning
Cuppens et al. [71]	Rule-BAC, Or-BAC	*CON*, *MIN*	N/A	FM—Reasoning
Adi et al. [16]	CA-BAC	*CON*	N/A	FM—Reasoning
Wang et al. [221]	Access Control	*CON*, *MIN*	N/A	FM—Reasoning
Sun et al. [211]	Access Control	*CON*	N/A	GM—Tree
Kolaczek [124]	RBAC	*CON*, *COR*	N/A	FM—Reasoning
Liu [135]	Network—Firewall	*MIN*, *COR*	*CIA*	MC—Decision Diagrams, FM—Reasoning
Gupta et al. [93]	RBAC	*CON*	N/A	FM—Reasoning
EXAM [131]	XACML	N/A	*SA*	MC—SAT Solver, MTBDD

(Continued)

Table 3. Continued

Publications	Policy Domain	Policy Quality	Policy Design	Method
Margrave [85]	XACML	*CON*	*CIA*	MC—MTBDD
FIREMAN [228]	Network	*CON*, *MIN*	N/A	MC—BDD
FLCheck [56]	RBAC	*CON*	N/A	MC—BDD
Pina Ros et al. [186]	XACML	*MIN*	N/A	MC—BDD
Hughes and Bultan [106]	XACML	*COR*	N/A	MC—Alloy, MC—SAT Solver
Jeffrey and Samak [116]	Network	*COR*	N/A	MC—SAT Solver
Alberti et al. [25, 26]	RBAC	*MIN*, *COR*	N/A	FM—Reasoning, MC—SMT Solver
MOHAWK [115]	RBAC	*COR*	N/A	MC—BDD, MC—SAT Solver
Jayaraman et al. [114]	Network	*CON*, *COR*	*CIA*	MC—SMT Solver
Mankai and Logrippo [154]	XACML	*CON*	N/A	MC—Alloy
Hassan et al. [101]	XACML	*CON*	N/A	MC—Alloy
Power et al. [187]	RBAC	*COR*	*CIA*	MC—Alloy
Karimi et al. [119], Karimi and Cowan [120]	RBAC, XACML	*CON*, *COR*	N/A	MC—Alloy, MC—SAT Solver, MC—BDD
Schaad and Moffett [198]	RBAC	*CON*	N/A	MC—Alloy
Haraty and Naous [99]	Network	*CON*	N/A	MC—Alloy
Guttman et al. [94]	Access Control	*COR*	N/A	MC—BDD
Tanvir and Tripathi [214]	RBAC	*CON*, *COM*, *COR*	N/A	MC—SAT Solver
AcPeg [231, 232]	XACML	*COM*, *COR*	N/A	MC—BDD
Ma et al. [152]	Access Control	*CON*, *COM*, *COR*	N/A	MC—SAT Solver
Hu and Ahn [103]	RBAC	*CON*, *COR*	N/A	MC—Alloy, MC—SAT Solver, FM—Reasoning
Hwang et al. [110]	XACML	*COR*	N/A	MC—SAT Solver, MC—BDD
PoliVer Koleini and Ryan [125]	Access Control	*COR*	N/A	FM—Reasoning, MC—BDD
Ngo et al. [182]	XACML	*MIN*, *COR*	N/A	MC—MIDD
Nelson et al. [180]	Network—SDN	*COR*	N/A	MC—Alloy
Mazzoleni et al. [164, 165]	XACML	N/A	*SA*	FM—Reasoning
Golnabi et al. [89]	Network—Firewall	*MIN*	N/A	DM—Association Rule Mining
Bauer et al. [41]	RBAC	*CON*, *COR*	N/A	DM - Association Rule Mining

(Continued)

Table 3. Continued

Publications	Policy Domain	Policy Quality	Policy Design	Method
Ma et al. [153]	RBAC	*COM*, *COR*	N/A	DM—Association Rule Mining
Marouf et al. [155, 156]	XACML	N/A	*PSS*	DM—Clustering
Ait El Hadj et al. [19]	XACML	*MIN*	N/A	DM—Clustering
Shaikh et al. [201, 202, 203]	Access Control	*CON*, *COM*, *MIN*	N/A	DM—Classification
Aqib and Shaikh [31]	Access Control	*CON*	N/A	DM—Classification
Schlegelmilch and Steffens [199]	RBAC	N/A	*PSS*	DM—Role Mining, DM—Clustering
Molloy et al. [172]	RBAC	N/A	*PSS*	DM—Role Mining
Dunlop et al. [80]	Management	*CON*	N/A	DM—Classification
Mukkamala et al. [179]	RBAC	*COR*	N/A	DM—Role Mining
Basile et al. [40]	Network—Firewall	*CON*, *MIN*	N/A	DM—Classification
Xu et al. [226]	Access Control	*COR*	*CIA*	GM—Graph
Staniford-Chen et al. [210]	RBAC	*COR*	N/A	GM—Graph
Alves and Fernández [27, 28]	RBAC	*COR*	N/A	GM—Graph
ProFact [14, 45]	RBAC	*CON*, *COM*, *MIN*, *REL*	N/A	GM—Tree, DM—Classification
Davy et al. [76]	Network	*CON*	N/A	GM—Tree
Al-Shaer et al. [21–24]	Network—Firewall	*CON*, *MIN*, *REL*, *COR*	N/A	GM—Tree
Sarna-Starosta and Stoller [197]	Access Control	*CON*, *COM*, *COR*	N/A	GM—Graph
Koch et al. [123]	LBAC	*CON*	N/A	GM—Graph
Russello et al. [193]	Access Control	*CON*	N/A	GM—Graph
Mohan et al. [171]	XACML	*CON*	N/A	GM—Tree
Huang et al. [105]	RBAC	*CON*	N/A	GM—Graph
Bravo et al. [53]	Access Control	*CON*	N/A	GM—Graph
Aqib and Shaikh [30]	Access Control	*CON*, *COR*	N/A	GM—Tree
Martin et al. [157–161]	XACML	*COR*	*CIA*	MT
Pretschner et al. [188]	RBAC	*COR*	N/A	MT
Mouelhi et al. [175, 176, 177, 178]	RBAC, XACML, OrBAC	*CON*, *COR*	N/A	MT
Masood et al. [162]	RBAC	*COR*	N/A	MT

(Continued)

Table 3. Continued

Publications	Policy Domain	Policy Quality	Policy Design	Method
Elrakaiby et al. [81]	RBAC	*COR*	*CIA*	MT
Le Traon et al. [130]	RBAC	*COR*	N/A	MT
Xu et al. [224]	XACML	*COR*	N/A	MT
Gorgias-B Spanoudakis et al. [209]	Access Control, Network—Firewall	*CON*	N/A	FM—Reasoning
Gouda and Liu [90, 91]	Network—Firewal	*CON*, *COM*, *MIN*	N/A	MC—Decision Diagram
Liu and Gouda [140, 142]	Network—Firewall	*MIN*	CIA	MC—Decision Diagram
Liu and Gouda [143]	Network—Firewall	*CON*	N/A	MC—Decision Diagram
Liu [133]	Network—Firewall	*COR*	N/A	MC—Decision Diagram
Liu et al. [141, 144, 146]	Network—Firewall	*MIN*	N/A	MC—Decision Diagram
Liu and Chen [136, 137]	Network—Firewall	*CON*, *COM*	N/A	MC—Decision Diagram
Liu et al. [138, 139]	XACML	*CON*, *COM*	N/A	MC—Decision Diagram
Liu et al. [147, 148]	Network—Firewall	*CON*, *COM*, *MIN*	N/A	MC—Decision Diagram
Hwang et al. [108, 109]	Network—Firewall	*COR*	CIA	MT
Chen et al. [63, 64]	Network—Firewall	*CON*, *COM*, *COR*	CIA	MC—Decision Diagram, MT
Khakpour and Liu [122], Liu and Khakpour [145]	Network—Firewall	*CON*, *COM*	N/A	MC—Decision Diagram, FM—Matrix based
Chen et al. [61, 62]	Network—Firewall	*CON*, *COM*, *MIN*	PSS	MC—Decision Diagram

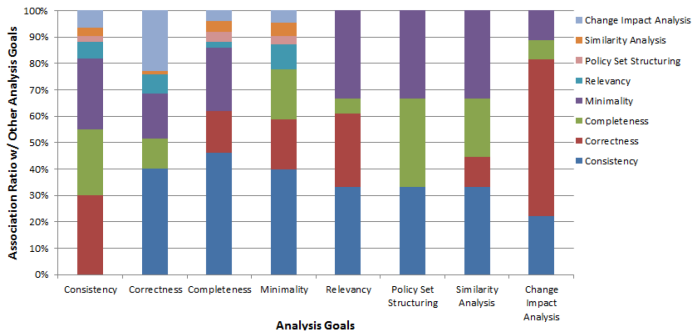


Fig. 4. Association among the analysis goals addressed by the policy analysis systems and frameworks described in the surveyed articles.

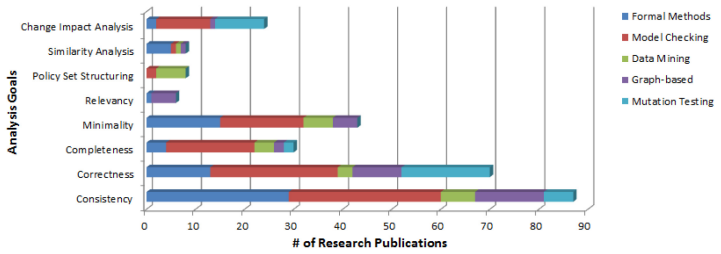


Fig. 5. The distribution of the analysis methods used for each analysis goal addressed by the policy analysis systems and frameworks described in the surveyed articles.

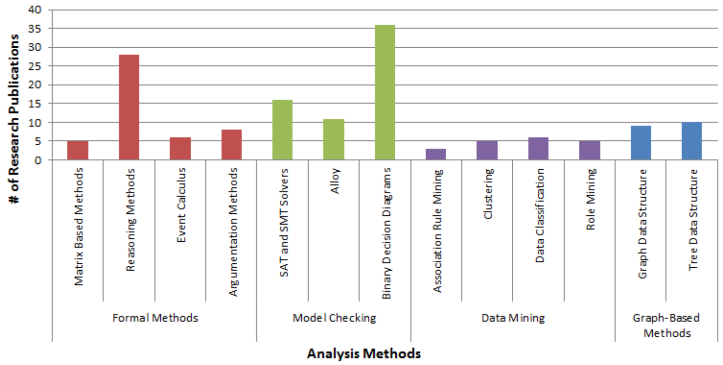


Fig. 6. The distribution of techniques per analysis methods used by the policy analysis systems and frameworks described in the surveyed articles.

and policy set structuring goals. In particular, the policy set analysis goal is associated with only three other goals: consistency, completeness, and minimality.

Figure 5 shows the distribution of the methods used for verifying each goal for the policy analysis systems described in the surveyed articles. In general, every goal is assessed by various methods. Consistency, correctness, minimality, and completeness are mainly evaluated by model checking and formal methods because these methods are comprehensive and easy to use. Hence, policy analysis approaches using model checking and formal methods can be a baseline or benchmark when developing new systems. Other analysis goals (i.e., change-impact analysis, relevancy, and policy set structuring) are analyzed using other techniques that suitably fit the nature of these goals. For example, change-impact analysis is evaluated using mutation testing since it is the most relevant method for applying various changes to the available set of policies in order to determine the impact on the quality of the analyzed policies.

Figure 6 shows the distribution of the various techniques for each analysis method used in the surveyed articles. In general, the distribution of the techniques among every method is almost uniform except for formal methods and model checking. For formal methods, the most-used technique is reasoning while the least-used technique is the matrix-based one. The reasoning methods are widely used because of their ability to generate analytical findings and results based on partial knowledge and observations of policies. The matrix-based methods are the least used because of their complexity and usage limitations. For model-checking methods, the most-used technique is the decision diagram–based technique while the least-used technique is the Alloy-based one. Decision diagrams are widely used because of their ability to represent all possible states of policies,

hence reducing the ambiguities that arise in policy enforcement systems, resulting in accurate and comprehensive analysis results.

7 RESEARCH DIRECTIONS

The area of methods, systems, and frameworks for policy analysis has been widely investigated, especially in the context of access control systems and network management systems. However, as our previous section has shown, there are no comprehensive systems able to support a variety of analysis with different goals. Most systems focus on some specific type of analysis with specific goals, for example, consistency analysis. It is thus clear that more general policy analysis systems are needed. In what follows, we discuss several relevant directions for analysis systems and methods.

7.1 New Goals for Policy Analysis

Policy analysis is an area in which there are still important open research directions specifically motivated by the deployment of policy-based management in the context of distributed systems consisting of autonomous intelligent devices. A first relevant research direction is related to the identification of analysis goals for such contexts. Two important goals are the assessment of enforceability and risk. Enforceability refers to an assessment of the feasibility and cost of enforcing a set of policies. For example, a policy whose enforcement requires access to top secret information may be very difficult to enforce in an insecure enforcement environment as the transfer of the top secret information into the enforcement system may not be possible. This means that this policy may be difficult to enforce and may require a specialized partitioned architecture that may be expensive. Assessing the risks arising from the use of specific policies is critical and non-trivial, as risks are usually application and context dependent. Further, the risk appetite of system owners could change. Therefore risk assessment methods are required to able to assess policies with respect to specific applications and also to able to continuously assess risks to deal with changing contexts and situations. Thus, policy changes may be required in response to such a changing risk and risk appetite.

7.2 New Methods for Policy Analysis

New methods for policy analysis ought to take into account policy goals of enforceability and risk and evaluate them in the presence of uncertainty or unpredictability of the context in which systems operate. Therefore they cannot be restricted to only traditional analysis tasks nor they can assume "perfect" inference about properties of the environment and of the system state. The new methods of analysis will need to be more stochastic in nature in both the representation of new policy goals and the analysis process. For example, analysis methods based on probabilistic model-checking techniques (e.g., PRISM [129]) could be used to support automated verification of quantitative analysis of enforceability and risk. Probabilistic model checking can support the analysis of several types of probabilistic models (e.g., discrete-time Markov chains, probabilistic automata, and probabilistic timed automata) with respect to properties expressed in probabilistic temporal logic (PCTL). Risk scenarios could be modeled using Markov chain (MC) models of stochastic (discrete) risk-related policy-driven scenarios, and probabilistic model checking could be used to evaluate expected risk implied by these models by expressing these goals using probability formalisms (e.g., PCTL [98]) and evaluating the models against these properties. A similar approach can be used for the analysis of the enforceability of policies. MC models are normally extended with notions of costs and rewards used to estimate resources and their use. The challenge is how to express policy specifications into these models for the purpose of analysis and what

types of properties to express in order to capture notions of risk and enforceability. A recently proposed technique that combines probabilistic and abductive inference for reasoning about most likely explanations of violations of given goals [216] could also provide an alternative underpinning framework for new methods of policy analysis. Such methods would constitute a natural extension of established state-of-the-art policy analysis techniques based on abductive inference to the context of uncertainty and risk. Both proposals would cover the case of static analysis of policies with respect to new classes of goals. Analysis methods based on runtime executions of autonomous intelligent systems could also be envisaged. These would rely on analysis of outcomes of enforced policies. Advanced machine-learning algorithms could be used to recognize patterns of risk and/or (lack of) enforceability of policies through past execution traces. Policies could then be analyzed with respect to these learned patterns and outcomes of the analysis could be used to inform ways to revise and adapt the policies. The trained machine-learning architectures could also be applied to new (unseen) situations to predict future goal violations.

7.3 New Analysis Frameworks for Distributed Systems

Another interesting research direction concerns policy analysis in large-scale distributed systems and refers to whether policy analysis (and possibly consequent policy evolution) must be carried centrally or locally at different subsystems. Ensuring that a set of policies is of high quality at a global level may be difficult if at all possible, as different portions of the system may be characterized by different access patterns and different contexts. On the other hand, carrying out policy analysis according to a distributed strategy so that policies are analyzed at different subsystems may result in policy analysis and evolution that are optimal with respect to local contexts but not optimal with respect to a more global level. A possible approach is to have a flexible analysis infrastructure able to support and possibly combine both approaches depending on the specific requirements of the system of interest. This is particularly relevant in dynamic network environments (e.g., coalitions and federated networks) that may involve network entities from different parties.

7.4 Next-Generation Policy-Based Management for Autonomous Devices

Finally, a newly emerging approach in the area of policy-based management is represented by the notion of generative policies [220] that has been proposed as the next-generation policy-based management approach for cognitive autonomous devices and IoT devices. In a generative policy framework, devices are given policy templates; they can autonomously instantiate and refine such templates according to their own missions and contexts. The generative policy framework will require novel analysis techniques to analyze policy templates and how these templates are instantiated by different devices in different contexts and for different missions (e.g., cognitive and distributed collaborative applications) and compare different instantiations to determine, for example, the optimal ones for specific contexts. Since the generative policy approach brings the agility of generating policies based on the contexts, an analysis of the change impact might be needed to assess the potential consequences and risks associated with the change. An efficient mechanism is required for conducting such a dynamic policy analysis for cognitive autonomous devices with limited storage and computation capabilities. Also, for IoT devices that have limited computing capabilities, some of the functions for policy generation may have to be delegated to more powerful devices and systems, such as edge servers. For those devices, an important issue is to analyze policies in order to determine the costs of policy enforcement with respect to energy and storage.

REFERENCES

[1] [n.d.]. Alloy: A Language & Tool for Relational Models. Retrieved January 27, 2018 from http://alloy.mit.edu/alloy/index.html.

[2] [n.d.]. Cbench: An OpenFlow Controller Benchmark. Retrieved July 20, 2018 from https://github.com/trema/cbench.

[3] [n.d.]. Extensible Access Control Markup Language (XACML) Version 3.0. Retrieved January 27, 2018 from http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html.

[4] [n.d.]. Gorgias-B. Retrieved January 27, 2018 from http://gorgiasb.tuc.gr/index.html.

[5] [n.d.]. List of SDN Activities. Retrieved January 27, 2018 from https://github.com/sdnds-tw/awesome-sdn.

[6] [n.d.]. NuSMV. Retrieved January 27, 2018 from http://nusmv.fbk.eu/.

[7] [n.d.]. OpenSMT. Retrieved January 27, 2018 from http://verify.inf.usi.ch/opensmt.

[8] [n.d.]. XSB. Retrieved January 27, 2018 from http://xsb.sourceforge.net.

[9] 2017. Authorization and Permissions in SQL Server. Retrieved January 27, 2018 from https://msdn.microsoft.com/en-us/library/bb669084(v=vs.110).aspx.

[10] 2018. Veryx Technologies. Retrieved June 23, 2018 from http://www.veryxtech.com/products/pktblaster-sdn-software-defined-network-test/.

[11] A . Abu Jabal and E. Bertino. 2016. QL-SimP: Query language for secure interoperable multi-granular provenance framework. In *CIC*. IEEE, 131–138.

[12] A. Abu Jabal and E. Bertino. 2016. SimP: Secure interoperable multi-granular provenance framework. In *e-Science*. IEEE, 270–275.

[13] A. Abu Jabal and E. Bertino. 2018. A comprehensive query language for provenance information. *International Journal of Cooperative Information Systems* 27, 3 (2018), 1850007–1850027.

[14] A. Abu Jabal, M. Davari, E. Bertino, C. Makaya, S. Calo, D. Verma, and C. Williams. 2018. ProFact: A Provenance-based Analytics Framework for Access Control Policies. (2018). Manuscript submitted for publication.

[15] A. T. Acree Jr. 1980. *On Mutation*. Ph.D. Dissertation. Georgia Institute of Technology, School of Information and Computer Science.

[16] K. Adi, Y. Bouzida, I. Hattak, L. Logrippo, and S. Mankovskii. 2009. Typing for conflict detection in access control policies. In *Proceedings of the 4th International Conference on E-Technologies*. Springer, 212–226.

[17] R. Agrawal, T. Imieliński, and A. Swami. 1993. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, Vol. 22. ACM, 207–216.

[18] R. Agrawal and R. Srikant. 1994. Fast algorithms for mining association rules. In *Proceedings of the 20th International Conference on Very Large Data Bases (VLDB'94)*, Vol. 1215. Morgan Kaufmann Publishers Inc., 487–499.

[19] M. Ait El Hadj, M. Ayache, Y. Benkaouz, A. Khoumsi, and M. Erradi. 2017. Clustering-based approach for anomaly detection in XACML policies. In *Proceedings of the 14th International Joint Conference on e-Business and Telecommunications (ICETE'17)*. SciTePress, 548–553.

[20] S. B. Akers. 1978. Binary decision diagrams. *IEEE Trans. Comput.* 6 (1978), 509–516.

[21] E. Al-Shaer and H. Hamed. 2003. Firewall policy advisor for anomaly discovery and rule editing. In *IM*. IEEE, 17–30.

[22] E. Al-Shaer, H. Hamed, R. Boutaba, and M. Hasan. 2005. Conflict classification and analysis of distributed firewall policies. *J-SAC* 23, 10 (2005), 2069–2084.

[23] E. Al-Shaer and H. H. Hamed. 2004. Modeling and management of firewall policies. *TNSM* 1, 1 (2004), 2–10.

[24] E. S. Al-Shaer and H. H. Hamed. 2004. Discovery of policy anomalies in distributed firewalls. In *INFOCOMM*, Vol. 4. IEEE, 2605–2616.

[25] F. Alberti, A. Armando, and S. Ranise. 2011. ASASP: Automated symbolic analysis of security policies. In *CADE*. Springer, 26–33.

[26] F. Alberti, A. Armando, and S. Ranise. 2011. Efficient symbolic automated analysis of administrative attribute-based RBAC-policies. In *ASIACCS*. ACM, 165–175.

[27] S. Alves and M. Fernández. 2015. A framework for the analysis of access control policies with emergency management. *ENTCS* 312 (2015), 89–105.

[28] S. Alves and M. Fernández. 2016. A graph-based framework for the analysis of access control policies. *Theoretical Computer Science (TCS)* 685 (2016), 3–22.

[29] A. Applebaum, K. N. Levitt, J. Rowe, and S. Parsons. 2012. Arguing about firewall policy. In *COMMA*, Vol. 245. 91–102.

[30] M. Aqib and R. A. Shaikh. 2014. An algorithm to detect inconsistencies in access control policies. In *Proceedings of the Intl. Conf. on Advances in Computing, Communication and Information Technology (CCIT'14)*. 171–175.

[31] M. Aqib and R. A. Shaikh. 2018. A tool for access control policy validation. *Journal of Internet Technology (JIT)* 19, 1 (2018), 157–166.

[32] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. 1997. Algebraic decision diagrams and their applications. *Formal Methods in System Design* 10, 2–3 (1997), 171–206.

[33] T. Ball, N. Bjørner, A. Gember, S. Itzhaky, A. Karbyshev, M. Sagiv, M. Schapira, and A. Valadarsky. 2014. VeriCon: towards verifying controller programs in software-defined networks. In *ACM Sigplan Notices*, Vol. 49. ACM, 282–293.
[34] A. Bandara, S. Calo, J. Lobo, E. Lupu, A. Russo, and M. Sloman. 2007. Toward a formal characterization of policy specification & analysis. In *Electronic Proceedings of the Annual Fall Meeting (AFM) of the International Technology Alliance (ICT)*.
[35] A. Bandara, E. Lupu, A. Russo, N. Dulay, M. Sloman, P. Flegkas, M. Charalambides, and G. Pavlou. 2005. Policy refinement for DiffServ quality of service management. In *IM*. IEEE, 469–482.
[36] A. K Bandara, A. Kakas, E. C. Lupu, and A. Russo. 2006. Using argumentation logic for firewall policy specification and analysis. In *DSOM*, Vol. 4269. Springer, 185–196.
[37] A. K. Bandara, A. C. Kakas, E. C. Lupu, and A. Russo. 2009. Using argumentation logic for firewall configuration management. In *IM*. IEEE, 180–187.
[38] A. K Bandara, E. C. Lupu, and A. Russo. 2003. Using event calculus to formalise policy specification and analysis. In *POLICY*. IEEE, 26–39.
[39] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. 2009. Satisfiability modulo theories. *Handbook of Satisfiability* 185 (2009), 825–885.
[40] C. Basile, A. Cappadonia, and A. Lioy. 2012. Network-level access control policy analysis and transformation. *TON* 20, 4 (2012), 985–998.
[41] L. Bauer, S. Garriss, and M. K. Reiter. 2011. Detecting and resolving policy misconfigurations in access-control systems. *TISSEC* 14, 1 (2011), 2.
[42] T. Bench-Capon. 2002. Value based argumentation frameworks. *arXiv preprint cs/0207059* (2002).
[43] S. Benferhat, R. El Baida, and F. Cuppens. 2003. A stratification-based approach for handling conflicts in access control. In *SACMAT*. ACM, 189–195.
[44] Y. Benkaouz, M. Erradi, and B. Freisleben. 2016. Work in progress: K-nearest neighbors techniques for ABAC policies clustering. In *ABAC*. ACM, 72–75.
[45] E. Bertino, A. Abu Jabal, S. Calo, C. Makaya, M. Touma, D. Verma, and C. Williams. 2017. Provenance-based analytics services for access control policies. In *SERVICES*. IEEE, 94–101.
[46] E. Bertino, A. Abu Jabal, S. Calo, D. Verma, Dinesh and C. Williams. 2018. The challenge of access control policies quality. *Journal of Data and Information Quality (JDIQ)* 10, 2 (2018), 6 pages.
[47] E. Bertino, P. A. Bonatti, and E. Ferrari. 2001. TRBAC: A temporal role-based access control model. *TISSEC* 4, 3 (2001), 191–233.
[48] E. Bertino, S. Calo, M. Touma, D. Verma, C. Williams, and B. Rivera. 2017. A cognitive policy framework for next-generation distributed federated systems: Concepts and research directions. In *ICDCS*. IEEE, 1876–1886.
[49] E. Bertino, G. Ghinita, and A. Kamra. 2011. Access control for databases: Concepts and systems. *Foundations and Trends® in Databases* 3, 1–2 (2011), 1–148.
[50] P. Besnard and A. Hunter. 2001. A logic-based theory of deductive arguments. *Artificial Intelligence* 128, 1–2 (2001), 203–235.
[51] G. Boella, J. Hulstijn, and L. van der Torre. 2005. Argument games for interactive access control. In *WI*. IEEE, 751–754.
[52] G. Boella, J. Hulstijn, and L. Van Der Torre. 2005. Argumentation for access control. In *Proc. of the 9th Congress of Italian Association for Artificial Intelligence*. Springer, 86–97.
[53] L. Bravo, J. Cheney, and I. Fundulaki. 2008. ACCOn: Checking consistency of XML write-access control policies. In *EDBT*. ACM, 715–719.
[54] T. A. Budd. 1980. Mutation Analysis of Program Test Data. Yale University.
[55] J. Catlett. 1991. Mega induction: A test flight. In *Machine Learning*. Elsevier, 596–599.
[56] A. Cau, H. Janicke, and B. Moszkowski. 2013. Verification and enforcement of access control policies. *Formal Methods in System Design* 43, 3 (2013), 450–492.
[57] B. Cestnik. 1987. Assistant 86: A knowledge-elicitation tool for sophisticated users. *Progress in Machine Learning* 62 (1987), 31–45.
[58] D. B. Chapman, E. D. Zwicky, and D. Russell. 1995. *Building Internet Firewalls*. O'Reilly & Associates, Inc.
[59] M. Charalambides, P. Flegkas, George Pavlou, Arosha K. Bandara, Emil C. Lupu, Alessandra Russo, N. Dulav, M. Sloman, and J. Rubio-Loyola. 2005. Policy conflict analysis for quality of service management. In *POLICY*. IEEE, 99–108.
[60] M. Charalambides, P. Flegkas, G. Pavlou, J. Rubio-Loyola, A. K. Bandara, E. C. Lupu, A. Russo, M. Sloman, and N. Dulay. 2006. Dynamic policy analysis and conflict resolution for DiffServ quality of service management. In *NOMS*. IEEE, 294–304.
[61] F. Chen, B. Bruhadeshwar, and A. X. Liu. 2011. Privacy-preserving cross-domain network reachability quantification. In *ICNP*. IEEE, 155–164.
[62] F. Chen, B. Bruhadeshwar, and A. X. Liu. 2013. Cross-domain privacy-preserving cooperative firewall optimization. *TON* 21, 3 (2013), 857–868.

[63] F. Chen, A. X. Liu, J. Hwang, and T. Xie. 2010. First step towards automatic correction of firewall policy faults. In *Proceedings of the 24th Large Installation System Administration Conference (LISA'10)*. USENIX Association, 75–90.
[64] F. Chen, A. X. Liu, J. Hwang, and T. Xie. 2012. First step towards automatic correction of firewall policy faults. *TAAS* 7, 2 (2012), 27–49.
[65] E. M. Clarke. 2008. The birth of model checking. In *25 Years of Model Checking, History, Achievements, Perspectives*, Vol. 5000. LNCS, Springer, Berlin, 1–26.
[66] D. Comaniciu and P. Meer. 2002. Mean shift: A robust approach toward feature space analysis. *TPAMI* 24, 5 (2002), 603–619.
[67] S. A. Cook. 1971. The complexity of theorem-proving procedures. In *STOC*. ACM, 151–158.
[68] T. H. Cormen. 2009. *Introduction to Algorithms*. MIT Press.
[69] R. Craven, J. Lobo, E. Lupu, J. Ma, A. Russo, M. Sloman, and A. Bandara. 2008. A formal framework for policy analysis. *Imperial College London, Technical Report* (2008).
[70] R. Craven, J. Lobo, J. Ma, A. Russo, E. Lupu, and A. Bandara. 2009. Expressive policy analysis with enhanced system dynamicity. In *ASIACCS*. ACM, 239–250.
[71] F. Cuppens, N. Cuppens-Boulahia, and M. B. Ghorbel. 2007. High level conflict management strategies in advanced access control models. *ENTCS* 186 (2007), 3–26.
[72] M. L. Damiani, E. Bertino, B. Catania, and P. Perlasca. 2007. GEO-RBAC: A spatially aware RBAC. *ACM Transactions on Information and System Security (TISSEC)* 10, 1 (2007), 2.
[73] N. Damianou, N. Dulay, E. C. Lupu, and MS Sloman. 2000. Ponder: A language for specifying security and management policies for distributed systems. Technical Report, Department of Computing, Imperial College, London.
[74] M. Davis, G. Logemann, and D. Loveland. 1962. A machine program for theorem-proving. *Commun. ACM* 5, 7 (1962), 394–397.
[75] M. Davis and H. Putnam. 1960. A computing procedure for quantification theory. *JACM* 7, 3 (1960), 201–215.
[76] S. Davy, B. Jennings, and J. Strassner. 2008. Efficient policy conflict analysis for autonomic network management. In *EASE*. IEEE, 16–24.
[77] L. De Moura and N. Bjørner. 2008. Z3: An efficient SMT solver. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08)*. Springer, 337–340.
[78] R. A. DeMillo, R. J. Lipton, and F. G. Sayward. 1978. Hints on test data selection: Help for the practicing programmer. *Computer* 11, 4 (1978), 34–41.
[79] P. M. Dung. 1995. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial Intelligence* 77, 2 (1995), 321–357.
[80] N. Dunlop, J. Indulska, and K. Raymond. 2002. Dynamic conflict detection in policy-based management systems. In *EDOC*. IEEE, 15–26.
[81] Y. Elrakaiby, T. Mouelhi, and Y. Le Traon. 2012. Testing obligation policy enforcement using mutation analysis. In *ICST*. IEEE, 673–680.
[82] E. A. Emerson. 2008. The beginning of model checking: A personal perspective. In *25 Years of Model Checking, History, Achievements, Perspectives*, Vol. 5000. LNCS, Springer, Berlin, 27–45.
[83] A. D. Ferguson, A. Guha, C. Liang, R. Fonseca, and S. Krishnamurthi. 2012. Hierarchical policies for software defined networks. In *HotSDN*. ACM, 37–42.
[84] D. F. Ferraiolo, R. Sandhu, S. Gavrila, D. R. Kuhn, and R. Chandramouli. 2001. Proposed NIST standard for role-based access control. *TISSEC* 4, 3 (2001), 224–274.
[85] K. Fisler, S. Krishnamurthi, L. A. Meyerovich, and M. C. Tschantz. 2005. Verification and change-impact analysis of access-control policies. In *ICSE*. IEEE, 196–205.
[86] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker. 2011. Frenetic: A network programming language. In *Proceeding of the 16th SIGPLAN ICFP*, Vol. 46. ACM, 279–291.
[87] M. Fujita, P. C. McGeer, and J. Yang. 1997. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal Methods in System Design* 10, 2–3 (1997), 149–169.
[88] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda. 2011. MIRAGE: A management tool for the analysis and deployment of network security policies. In *DPM*. Springer, 203–215.
[89] K. Golnabi, R. K. Min, L. Khan, and E. Al-Shaer. 2006. Analysis of firewall policy rules using data mining techniques. In *NOMS*. IEEE, 305–315.
[90] M. G. Gouda and A. X. Liu. 2007. Structured firewall design. *Computer Networks* 51, 4 (2007), 1106–1120.
[91] M. G. Gouda and X. Liu. 2004. Firewall design: Consistency, completeness, and compactness. In *ICDCS*. IEEE, 320–327.
[92] D. P. Guelev, M. Ryan, and P. Schobbens. 2004. Model-checking access control policies. In *ISC*, Vol. 3225. Springer, 219–230.
[93] P. Gupta, S. D. Stoller, and Z. Xu. 2014. Abductive analysis of administrative policies in rule-based access control. *ITDSC* 11, 5 (2014), 412–424.

[94] J. D. Guttman, A. L. Herzog, J. D. Ramsdell, and C. W. Skorupka. 2005. Verifying information flow goals in security-enhanced Linux. *JCS* 13, 1 (2005), 115–134.
[95] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. 2009. The WEKA data mining software: An update. *SIGKDD Explorations Newsletter* 11, 1 (2009), 10–18.
[96] J. Y. Halpern and V. Weissman. 2008. Using first-order logic to reason about policies. *TISSEC* 11, 4 (2008), 21.
[97] J. Han, J. Pei, and Y. Yin. 2000. Mining frequent patterns without candidate generation. In *SIGMOD*, Vol. 29. ACM, 1–12.
[98] H. Hansson and B. Jonsson. 1994. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6, 5 (1994), 512–535.
[99] R. A. Haraty and M. Naous. 2013. Modeling and validating the clinical information systems policy using alloy. In *HIS*. Springer, 1–17.
[100] S. M. Hasani and N. Modiri. 2013. Criteria specifications for the comparison and evaluation of access control models. *IJICS* 5, 5 (2013), 19.
[101] W. Hassan, L. Logrippo, and M. Mankai. 2005. Validating access control policies with alloy. In *Proceedings of the Workshop on Practice and Theory of Access Control Technologies*. 17–22.
[102] T. L Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker. 2009. Practical declarative network management. In *SIGCOMM Workshop on Research on Enterprise Networking*. ACM, 1–10.
[103] H. Hu and G. Ahn. 2008. Enabling verification and conformance testing for access control model. In *SACMAT*. ACM, 195–204.
[104] H. Hu, W. Han, G. Ahn, and Z. Zhao. 2014. FLOWGUARD: Building robust firewalls for software-defined networks. In *HotSDN*. ACM, 97–102.
[105] C. Huang, J. Sun, X. Wang, and Y. Si. 2009. Inconsistency management of role base access control policy. In *EBISS*. IEEE, 1–5.
[106] G. Hughes and T. Bultan. 2008. Automated verification of access control policies using a SAT solver. *STTT* 10, 6 (2008), 503–520.
[107] S. Hussain. 2008. *Mutation Clustering*. Master's thesis. King's College London.
[108] J. Hwang, T. Xie, F. Chen, and A. X. Liu. 2008. Systematic structural testing of firewall policies. In *SRDS*. IEEE, 105–114.
[109] J. Hwang, T. Xie, F. Chen, and A. X. Liu. 2012. Systematic structural testing of firewall policies. *TNSM* 9, 1 (2012), 1–11.
[110] J. Hwang, T. Xie, V. Hu, and M. Altunay. 2010. ACPT: A tool for modeling and verifying access control policies. In *POLICY*. IEEE, 40–43.
[111] D. Jackson. 2002. Alloy: A lightweight object modelling notation. *TOSEM* 11, 2 (2002), 256–290.
[112] D. Jackson, I. Schechter, and H. Shlyahter. 2000. Alcoa: The alloy constraint analyzer. In *ICSE*. ACM, 730–733.
[113] M. Jarschel, F. Lehrieder, Z. Magyari, and R. Pries. 2012. A flexible OpenFlow-Controller benchmark. In *EWSDN*. IEEE, 48–53.
[114] K. Jayaraman, N. Bjørner, G. Outhred, and C. Kaufman. 2014. Automated analysis and debugging of network connectivity policies. Technical Report, Microsoft Research.
[115] K. Jayaraman, V. Ganesh, M. Tripunitara, M. Rinard, and S. Chapin. 2011. Automatic error finding in access-control policies. In *CCS*. ACM, 163–174.
[116] A. Jeffrey and T. Samak. 2009. Model checking firewall policy configurations. In *POLICY*. IEEE, 60–67.
[117] Y. Jia and M. Harman. 2008. Constructing subtle faults using higher order mutation testing. In *SCAM*. IEEE, 249–258.
[118] Y. Jia and M. Harman. 2011. An analysis and survey of the development of mutation testing. *TSE* 37, 5 (2011), 649–678.
[119] V. R. Karimi, P. S. C. Alencar, and D. D. Cowan. 2017. A formal modeling and analysis approach for access control rules, policies, and their combinations. *IJIS* 16, 1 (2017), 43–74.
[120] V. R. Karimi and D. D. Cowan. 2009. Verification of access control policies for REA business processes. In *COMPSAC*, Vol. 2. IEEE, 422–427.
[121] L. Kaufman and P. J. Rousseeuw. 2009. *Finding Groups in Data: An Introduction to Cluster Analysis*. Vol. 344. John Wiley & Sons.
[122] A. R. Khakpour and A. X. Liu. 2010. Quantifying and querying network reachability. In *ICDCS*. IEEE, 817–826.
[123] M. Koch, L. V. Mancini, and F. Parisi-Presicce. 2001. On the specification and evolution of access control policies. In *SACMAT*. ACM, 121–130.
[124] G. Kolaczek. 2003. Specification and verification of constraints in role based access control. In *WET ICE*. IEEE, 190–195.
[125] M. Koleini and M. Ryan. 2011. A knowledge-based verification method for dynamic access control policies. In *ICFEM*. Springer, 243–258.
[126] V. Kolovski, J. Hendler, and B. Parsia. 2007. Analyzing web access control policies. In *WWW*. ACM, 677–686.

[127] D. Kreutz, F. M. V. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig. 2015. Software-defined networking: A comprehensive survey. *Proc. IEEE* 103, 1 (2015), 14–76.
[128] M. Kumar and R. E. Newman. 2006. STRBAC-An approach towards spatio-temporal role-based access control. In *Proceedings of the Third IASTED International Conference on Communication, Network, and Information Security (CNIS)*. IASTED/ACTA Press, 150–155.
[129] M. Kwiatkowska, G. Norman, and D. Parker. 2011. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*. Springer, 585–591.
[130] Y. Le Traon, T. Mouelhi, and B. Baudry. 2007. Testing security policies: Going beyond functional testing. In *ISSRE*. IEEE, 93–102.
[131] D. Lin, P. Rao, E. Bertino, N. Li, and J. Lobo. 2010. EXAM: A comprehensive environment for the analysis of access control policies. *IJICS* 9, 4 (2010), 253–273.
[132] D. Lin, P. Rao, E. Bertino, and J. Lobo. 2007. An approach to evaluate policy similarity. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT'07)*. ACM, 1–10.
[133] Alex X. Liu. 2009. Firewall policy verification and troubleshooting. *Computer Networks* 53, 16 (2009), 2800–2809.
[134] A. X. Liu. 2010. Complete redundancy removal for packet classifiers in TCAMs. *TPDS* 21, 4 (2010), 424–437.
[135] A. X. Liu. 2012. Firewall policy change-impact analysis. *TOIT* 11, 4 (2012), 15:1–15:24.
[136] A. X. Liu and F. Chen. 2008. Collaborative enforcement of firewall policies in virtual private networks. In *PODC*. ACM, 95–104.
[137] A. X. Liu and F. Chen. 2011. Privacy preserving collaborative enforcement of firewall policies in virtual private networks. *TPDS* 22, 5 (2011), 887–895.
[138] A. X. Liu, F. Chen, J. Hwang, and T. Xie. 2008. XEngine: A fast and scalable XACML policy evaluation engine. In *SIGMETRICS*, Vol. 36. ACM, 265–276.
[139] A. X. Liu, F. Chen, J. Hwang, and T. Xie. 2011. Designing fast and scalable XACML policy evaluation engines. *IEEE Trans. Comput.* 60, 12 (2011), 1802–1817.
[140] A. X. Liu and M. G. Gouda. 2004. Diverse firewall design. In *DSN*. IEEE Computer Society, 595.
[141] A. X. Liu and M. G. Gouda. 2005. Complete redundancy detection in firewalls. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 193–206.
[142] A. X. Liu and M. G. Gouda. 2008. Diverse firewall design. *TPDS* 19, 9 (2008), 1237–1251.
[143] A. X. Liu and M. G Gouda. 2009. Firewall policy queries. *TPDS* 20, 6 (2009), 766–777.
[144] A. X. Liu and M. G. Gouda. 2010. Complete redundancy removal for packet classifiers in TCAMs. *TPDS* 21, 4 (2010), 424–437.
[145] A. X. Liu and A. R. Khakpour. 2013. Quantifying and verifying reachability for access controlled networks. *ToN* 21, 2 (2013), 551–565.
[146] A. X. Liu, C. R. Meiners, and Y. Zhou. 2008. All-match based complete redundancy removal for packet classifiers in TCAMs. In *INFOCOM*. IEEE, 111–115.
[147] A. X. Liu, E. Torng, and C. R. Meiners. 2008. Firewall compressor: An algorithm for minimizing firewall policies. In *INFOCOM*. IEEE, 176–180.
[148] A. X. Liu, E. Torng, and C. R. Meiners. 2011. Compressing network access control lists. *TPDS* 22, 12 (2011), 1969–1977.
[149] S. Lloyd. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* 28, 2 (1982), 129–137.
[150] P. Loscocco and S. Smalley. 2001. Integrating flexible support for security policies into the Linux operating system. In *FREENIX Track: 2001 USENIX Annual Technical Conference*. 29–42.
[151] E. C. Lupu and M. Sloman. 1999. Conflicts in policy-based distributed systems management. *TSE* 25, 6 (1999), 852–869.
[152] J. Ma, D. Zhang, G. Xu, and Y. Yang. 2010. Model checking based security policy verification and validation. In *ISA*. IEEE, 1–4.
[153] X. Ma, R. Li, Z. Lu, and W. Wang. 2012. Mining constraints in role-based access control. *Mathematical and Computer Modelling* 55, 1 (2012), 87–96.
[154] M. Mankai and L. Logrippo. 2005. Access control policies: Modeling and validation. In *NOTERE*. 85–91.
[155] S. Marouf, M. Shehab, A. Squicciarini, and S. Sundareswaran. 2009. Statistics & clustering based framework for efficient XACML policy evaluation. In *POLICY*. IEEE, 118–125.
[156] S. Marouf, M. Shehab, A. Squicciarini, and S. Sundareswaran. 2011. Adaptive reordering and clustering-based framework for efficient XACML policy evaluation. *TSC* 4, 4 (2011), 300–313.
[157] E. Martin. 2007. Testing and analysis of access control policies. In *ICSE*. IEEE Computer Society, 75–76.
[158] E. Martin, J. Hwang, T. Xie, and V. Hu. 2008. Assessing quality of policy properties in verification of access control policies. In *ACSAC*. IEEE, 163–172.
[159] E. Martin and T. Xie. 2007. Automated test generation for access control policies via change-impact analysis. In *SESS*. IEEE Computer Society, 5.

[160] E. Martin and T. Xie. 2007. A fault model and mutation testing of access control policies. In *WWW*. ACM, 667–676.
[161] E. Martin, T. Xie, and T. Yu. 2006. Defining and measuring policy coverage in testing access control policies. In *ICICS*, Vol. 6. Springer, 139–158.
[162] A. Masood, A. Ghafoor, and A. Mathur. 2006. *Scalable and Effective Test Generation for Access Control Systems that Employ RBAC Policies*. Technical Report. SERC-TR-285, Purdue University.
[163] A. P. Mathur. 1991. Performance, effectiveness, and reliability issues in software testing. In *COMPSAC*. IEEE, 604–605.
[164] P. Mazzoleni, E. Bertino, B. Crispo, and S. Sivasubramanian. 2006. XACML policy integration algorithms: Not to be confused with XACML policy combination algorithms!. In *SACMAT*. ACM, 219–227.
[165] P. Mazzoleni, B. Crispo, S. Sivasubramanian, and E. Bertino. 2008. XACML policy integration algorithms. *TISSEC* 11, 1 (2008), 4.
[166] P. McDaniel and A. Prakash. 2006. Methods and limitations of security policy reconciliation. *TISSEC* 9, 3 (2006), 259–291.
[167] N. McKeown. 2011. How SDN Will Shape Networking. Retrieved January 27, 2018 from http://www.youtube.com/watch?v=c9-K5OqYgA.
[168] N. McKeown, T. Anderson, and H. Balakrishna. 2008. OpenFlow: Enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.* 38, 2 (2008), 69–74.
[169] K. L. McMillan. 1993. Symbolic model checking. In *Symbolic Model Checking*. Springer, 25–60.
[170] J. D. Moffett and M. S. Sloman. 1994. Policy conflict analysis in distributed system management. *JOCEC* 4, 1 (1994), 1–22.
[171] A. Mohan, D. M. Blough, T. Kurc, A. Post, and J. Saltz. 2011. Detection of conflicts and inconsistencies in taxonomy-based authorization policies. In *BIBM*. IEEE, 590–594.
[172] I. Molloy, H. Chen, T. Li, Q. Wang, N. Li, E. Bertino, S. Calo, and J. Lobo. 2010. Mining roles with multiple objectives. *TISSEC* 13, 4 (2010), 36.
[173] C. Monsanto, N. Foster, R. Harrison, and D. Walker. 2012. A compiler and run-time system for network programming languages. In *SIGPLAN-SIGACT - POPL*, Vol. 47. ACM, 217–230.
[174] C. Monsanto, J. Reich, N. Foster, J. Rexford, and D. Walker. 2013. Composing software-defined networks. In *USENIX NSDI*. ACM, 21–14.
[175] T. Mouelhi, F. Fleurey, and B. Baudry. 2008. A generic metamodel for security policies mutation. In *ICST*. IEEE, 278–286.
[176] T. Mouelhi, F. Fleurey, B. Baudry, and Y. Le Traon. 2008. A model-based framework for security policy specification, deployment and testing. In *MoDELS*. Springer, 537–552.
[177] T. Mouelhi, Y. Le Traon, and B. Baudry. 2007. Mutation analysis for security tests qualification. In *TAICPART-MUTATION*. IEEE, 233–242.
[178] T. Mouelhi, Y. Le Traon, and B. Baudry. 2009. Transforming and selecting functional test cases for security policy testing. In *ICST*. IEEE, 171–180.
[179] R. Mukkamala, V. Kamisetty, and P. Yedugani. 2009. Detecting and resolving misconfigurations in role-based access control. *Information Systems Security* 5905 (2009), 318–325.
[180] T. Nelson, A. D. Ferguson, M. J. G. Scheer, and S. Krishnamurthi. 2014. Tierless programming and reasoning for software-defined networks. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI'14)*, Vol. 14. USENIX Association, 519–531.
[181] A. Y. Ng, M. I. Jordan, and Y. Weiss. 2002. On spectral clustering: Analysis and an algorithm. In *Proceedings of Advances in Neural Information Processing Systems (NIPS'02)*. MIT Press, 849–856.
[182] C. Ngo, Y. Demchenko, and C. de Laat. 2015. Decision diagrams for XACML policy evaluation and management. *Computers & Security* 49 (2015), 1–16.
[183] Q. Ni, E. Bertino, J. Lobo, C. Brodie, C. Karat, J. Karat, and A. Trombeta. 2010. Privacy-aware role-based access control. *TISSEC* 13, 3 (2010), 24.
[184] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *JMLR* 12, Oct (2011), 2825–2830.
[185] L. Perrussel, S. Doutre, J. Thévenin, and P. McBurney. 2007. A persuasion dialog for gaining access to information. In *ArgMAS*. Springer, 63–79.
[186] S. Pina Ros, M. Lischka, and F. Gómez Mármol. 2012. Graph-based XACML evaluation. In *SACMAT*. ACM, 83–92.
[187] D. J. Power, M. Slaymaker, and A. Simpson. 2011. Conformance checking of dynamic access control policies. In *ICFEM*. Springer, 227–242.
[188] A. Pretschner, T. Mouelhi, and Y. Le Traon. 2008. Model-based tests for access control policies. In *ICST*. IEEE, 338–347.
[189] J. R. Quinlan. 1986. Induction of decision trees. *Machine Learning* 1, 1 (1986), 81–106.
[190] J. R. Quinlan. 2014. *C4. 5: Programs for Machine Learning*. Elsevier.

[191] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. 1991. A model of authorization for next-generation database systems. *TODS* 16, 1 (1991), 88–131.
[192] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo. 2009. An algebra for fine-grained integration of XACML policies. In *SACMAT*. ACM, 63–72.
[193] G. Russello, C. Dong, and N. Dulay. 2007. Authorisation and conflict resolution for hierarchical domains. In *POLICY*. IEEE, 201–210.
[194] A. Russo, R. Miller, B. Nuseibeh, and J. Kramer. 2002. An abductive approach for analysing event-based requirements specifications. In *ICLP*. Springer, 22–37.
[195] R. Sandhu, V. Bhamidipati, and Q. Munawer. 1999. The ARBAC97 model for role-based administration of roles. *ISSEC* 2, 1 (1999), 105–135.
[196] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. 1996. Role-based access control models. *Computer* 29, 2 (1996), 38–47.
[197] B. Sarna-Starosta and S. D. Stoller. 2004. Policy analysis for security-enhanced Linux. In *Proceedings of the 2004 Workshop on Issues in the Theory of Security (WITS'04)*. ACM, 1–12.
[198] A. Schaad and J. D. Moffett. 2002. A lightweight approach to specification and analysis of role-based access control extensions. In *SACMAT*. ACM, 13–22.
[199] J. Schlegelmilch and U. Steffens. 2005. Role mining with ORCA. In *SACMAT*. ACM, 168–176.
[200] B. Selman, H. A. Kautz, B. Cohen, et al. 1993. Local search strategies for satisfiability testing. *Cliques, Coloring, and Satisfiability* 26 (1993), 521–532.
[201] R. A. Shaikh, K. Adi, and L. Logrippo. 2017. A data classification method for inconsistency and incompleteness detection in access control policy sets. *IJIS* 16, 1 (2017), 91–113.
[202] R. A. Shaikh, K. Adi, L. Logrippo, and S. Mankovski. 2010. Detecting incompleteness in access control policies using data classification schemes. In *ICDIM*. IEEE, 417–422.
[203] R. A. Shaikh, K. Adi, L. Logrippo, and S. Mankovski. 2010. Inconsistency detection method for access control policies. In *IAS*. IEEE, 204–209.
[204] S. Shenker, M. Casado, T. Koponen, N. McKeown, et al. 2011. The future of networking, and the past of protocols. *Open Networking Summit* 20 (2011), 1–30.
[205] S. Shin, V. Yegneswaran, P. Porras, and G. Gu. 2013. Avant-guard: Scalable and vigilant switch flow management in software-defined networks. In *CCS*. ACM, 413–424.
[206] B. Shneiderman and A. Aris. 2006. Network visualization by semantic substrates. *TVCG* 12, 5 (2006), 733–740.
[207] R. Sibson. 1973. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal* 16, 1 (1973), 30–34.
[208] F. Somenzi. 1998. CUDD: CU decision diagram package release 2.3.0. *University of Colorado at Boulder* (1998).
[209] N. I. Spanoudakis, A. C. Kakas, and P. Moraitis. 2016. Gorgias-B: Argumentation in practice. In *COMMA*. IEEE, 477–478.
[210] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, J. Hoagland, K. Levitt, C. Wee, R. Yip, and D. Zerkle. 1996. GrIDS-a graph based intrusion detection system for large networks. In *Proceedings of the 19th National Information Systems Security Conference (NISSC'96)*. Defense Technical Information Center, 361–370.
[211] L. Sun, H. Wang, X. Tao, Y. Zhang, and J. Yang. 2011. Privacy preserving access control policy and algorithms for conflicting problems. In *TrustCom*. IEEE, 250–257.
[212] H. Takabi and J. B. D. Joshi. 2010. StateMiner: An efficient similarity-based approach for optimal mining of role hierarchy. In *SACMAT*. ACM, 55–64.
[213] P. Tan, M. Steinbach, A. Karpatne, and V. Kumar. 2006. *Introduction to Data Mining*. Pearson Education India.
[214] A. Tanvir and A. R. Tripathi. 2003. Static verification of security requirements in role based CSCW systems. In *SACMAT*. ACM, 196–203.
[215] F. Turkmen, J. den Hartog, S. Ranise, and N. Zannone. 2015. Analysis of XACML policies with SMT. In *Proceedings of the International Conference on Principles of Security and Trust (POST'15)*. Springer, 115–134.
[216] C. R. Turliuc, L. Dickens, A. Russo, and K. Broda. 2016. Probabilistic abductive logic programming using Dirichlet priors. *International Journal of Approximate Reasoning* 78 (2016), 223–240.
[217] J. Vaidya, V. Atluri, and Q. Guo. 2007. The role mining problem: Finding a minimal descriptive set of roles. In *SACMAT*. ACM, 175–184.
[218] J. Vaidya, V. Atluri, and J. Warner. 2006. RoleMiner: Mining roles using subset enumeration. In *CCS*. ACM, 144–153.
[219] F. H. Van Eemeren and R. Grootendorst. 2004. *A Systematic Theory of Argumentation: The Pragma-Dialectical Approach*. Vol. 14. Cambridge University Press.
[220] D. Verma, S. Calo, S. Chakraborty, E. Bertino, C. Williams, J. Tucker, and B. Rivera. 2017. Generative policy model for autonomic management. In *DAIS*. IEEE, 4–8.

[221] Y. Wang, H. Zhang, X. Dai, and J. Liu. 2010. Conflicts analysis and resolution for access control policies. In *ICITIS*. IEEE, 264–267.
[222] I. H. Witten, E. Frank, M. A. Hall, and C. J. Pal. 2016. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann.
[223] B. Wu, X. Chen, Y. Zhang, and X. Dai. 2009. An extensible intra access control policy conflict detection algorithm. In *CIS*, Vol. 1. IEEE, 483–488.
[224] D. Xu, Z. Wang, S. Peng, and N. Shen. 2016. Automated fault localization of XACML policies. In *SACMAT*. ACM, 137–147.
[225] W. Xu, M. Shehab, and G. Ahn. 2008. Visualization based policy analysis: Case study in SELinux. In *SACMAT*. ACM, 165–174.
[226] W. Xu, X. Zhang, and G. Ahn. 2009. Towards system integrity protection with graph-based policy analysis. In *(IFIP) (WG) 11.3 Working Conference Data and Applications Security*. Springer, 65–80.
[227] Z. Xu and S. D. Stoller. 2013. Mining attribute-based access control policies from RBAC policies. In *CEWIT*. IEEE, 1–6.
[228] L. Yuan, H. Chen, J. Mai, C. Chuah, Z. Su, and P. Mohapatra. 2006. Fireman: A toolkit for firewall modeling and analysis. In *S&P*. IEEE, 199–213.
[229] M. J. Zaki. 2000. Scalable algorithms for association mining. *TKDE* 12, 3 (2000), 372–390.
[230] L. Zhang and S. Malik. 2002. The quest for efficient Boolean satisfiability solvers. In *CADE*. Springer, 313–331.
[231] N. Zhang, M. Ryan, and D. P. Guelev. 2005. Evaluating access control policies through model checking. In *ISC*. Springer, 446–460.
[232] N. Zhang, M. Ryan, and D. P. Guelev. 2008. Synthesising verified access control systems through model checking. *JCS* 16, 1 (2008), 1–61.

Received February 2018; revised July 2018; accepted September 2018