JSON Schema for Attribute-based Access Control

Gregory Linklater[†], Christian Smith[‡], James Connan[†], Alan Herbert[†], Barry Irwin[†]
[†]*Rhodes University, Grahamstown, Eastern Cape, South Africa*
[1]`g12l4025@campus.ru.ac.za`
[3]`j.connan@ru.ac.za`
[4]`a.herbert@ru.ac.za`
[5]`b.irwin@ru.ac.za`
[‡]*MIT Connection Science, Cambridge, Massachusetts, USA*
[2]`csmth@mit.edu`

***Abstract*—Attribute-Based Access Control (ABAC) is an access control model where authorization for an action on a resource is determined by evaluating attributes of the subject, resource (object) and environment. The attributes are evaluated against arbitrarily complex boolean rules. ABAC rule languages are often based on serializable object modeling and schema languages as in the case of XACML which is based on XML Schema. XACML is a standard by OASIS, initially published in 2003 and updated in 2005 and 2013 and is the current *de facto* standard for ABAC. While a JSON profile for XACML exists, it is simply a compatibility layer for using JSON in XACML which caters to the XML object model paradigm, as opposed to the JSON object model paradigm. This research proposes JSON Schema as a modeling language that caters to the JSON object model paradigm from which to base an ABAC rule language and demonstrates its viability for the task by comparison against the features provided to XACML by XML Schema.**

***Index Terms*—JSON Schema; Access Control; Authorization; Security; Attribute-based Access Control; ABAC; Network Security**

I. Introduction

Access control is a mechanism for ensuring that limited or otherwise sensitive resources are shared only with those who are authorized to access or use them. Many access control models exist, however the topic of this research is focused on ABAC. ABAC architectures and rule languages are based on schema definition and object modeling languages. Mature languages such as XML and XML Schema [1], [2] have an existing ABAC standard, however the JavaScript Object Notation (JSON) does not.

A. Problem Statement

The current *de facto* standard for ABAC is the eXtensible Access Control Markup Language (XACML). XACML is used to secure REST services, physical barriers, databases and network resources as well as prevent loss of data through email, removable media and printing through subsequent additional profiles [3]. XACML is based on and caters to the XML and XML Schema object model paradigm [4]. A JSON standard does exist for XACML [5] however this is a compatibility layer which does not adequately provide for the object model paradigm of JSON [6].

B. Research Goal

JSON Schema [7], [8] is a schema definition and modeling language for JSON that was designed for the JSON object model paradigm. This research proposes JSON Schema as a modeling language from which to base an ABAC rule language and demonstrates its viability for the task by comparison against the features provided to XACML by XML Schema.

II. Access Control

More than half of recorded data breaches are being perpetrated by members internal to the organisation according to the respondents of the FBI's Crime Scene Investigation (CSI) Cybercrime Survey of 2009 [9]. This suggests that the perimeter security model is no longer sufficient to adequately protect resources.

Access control enables the implementation of concepts such as separation of duties and least privilege in security policy. Logging and auditing provide the checks and balances necessary to prove due diligence and due care in a court of law [10], [11]. Identity providers and ABAC are capable of providing the means to control access to any network or domain-controlled resource in as fine-grained or as broad a manner that is required by the resource owner. These together provide a method of mitigating risk from internal threats [12].

A. Access Control Models

Popular access control models include role– and identity–based access control which have the common problem of limited scalability. These models are effective up to a limited number of users before having to be replaced. This inhibits an organisation's growth unless the access control model is replaced, which carries a high cost. Of all available access control models, ABAC is considered to be the most versatile as it carries few limitations and is scalable with negligible administrative overhead [12], [13].

ABAC finds itself in a similar situation to Role-Based Access Control (RBAC) found itself in pre-1992 where it was not authoritatively defined despite literature on the topic existing for decades prior. To date, the most authoritative definition of ABAC can be found in the USA's National

Institute of Science and Technology (NIST) guide to ABAC [14]. The guide defines ABAC as follows:

> *"An access control method where subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions."* [14]

Models such as role– and lattice– based access control have features that are aligned with the original discretionary and mandatory access control models respectively or a combination thereof and carry the same limitations. Unlike these, ABAC can implement arbitrarily complex boolean rules for access, based on attributes of the subject, resource or environment; these are discussed further in Section III. This allows one to implement *any* and *multiple* of the models or patterns preceding it simultaneously, by modifying access control rules. [12], [14]

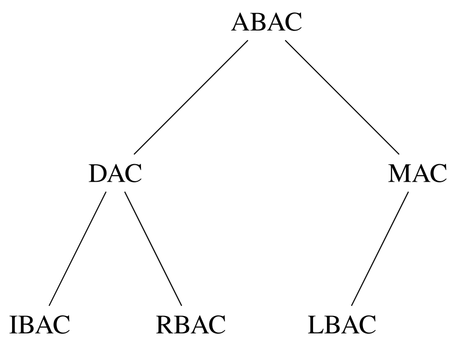

Fig. 1. Non-Exhaustive Access Control Model Hierarchy

ABAC is the superset of all access control models as it allows access control to be enacted based on any arbitrary set of attributes from a trustworthy source. The relationship between a non-exhaustive list of access control models is depicted in Figure 1. The derivative access control models share two things in common:

1) They are a subset of ABAC as they enact access control based on a particular subset of attributes. In the case of role–, lattice– and identity– based access control, these are the role, security clearance tag and unique identifier of the subject attributes respectively [14].
2) They group authorizations to execute actions on resources based on these attributes.

ABAC does not share the concept of grouping authorizations in order to ease the administrative process of authorizing subjects to complete actions. Instead, ABAC defines rules that are applied per action per resource. These rules and subsections of these rules can be reused, if desired, for multiple actions on multiple resources and administration of access instead happens at the level of the attributes of the subject. In order to grant or revoke access to these actions, one must simply modify the attributes of the subject to immediately reflect their new status.

III. ABAC Rules

The functionality that defines ABAC is the ability to arrive at an access decision by evaluating boolean logic rules against a set of attributes pertaining to the subject, resource and environment [13]. ABAC rules are intended to provide an abstraction for the end-user that allows them to define conditions for access without requiring bespoke program logic. These rules are typically expressed using language specifications derived from data modeling languages which feature boolean validation logic embedded in the model itself. Additionally, any ABAC rule language needs to be functionally complete — as discussed in Section III-B — in order to evaluate any combination of complex boolean logic.

A. Model Languages for ABAC Rules

Modeling languages, such as XML Schema, have provided a logical starting point for an ABAC rule language. XML already interacts with data at the level required by ABAC in fulfilling its bespoke purpose. The following functionality was determined to be of interest to the topic:

1) *Model*: The data model or schema specifies the form and metadata of data as well as metadata about the model itself. With a model, one can define and expect the data specified by a model in a given document that claims to conform to that model. Complete model conformance is not guaranteed as the data provided could have missing or invalid data; this is solved by validation.
2) *Data Interaction*: Modeling languages provide a one-to-one mapping with data that conforms to the defined model. The metadata of the data in a particular field can be found by looking up the name of the field in the model and *vise versa*.
3) *Validation*: Successful model validation provides the guarantee that the data specified by the model is present and valid where the model alone does not. If the validation against the schema fails, by definition the data is not valid. Model validation is often either provided within the specification of the model language itself or in subsequent derivative specifications.
4) *Serialization*: Although models are — by definition — metadata, they are still data and need to be created, stored, retrieved and transmitted like data. For this reason, modeling languages tend to be based on the serializable data representation language that they serve to create models for, in order to inherit the serializable property.

Although there are differences in intended use, ABAC rules and models are functionally similar. XML Schema has been successfully used as the base modeling language for the XACML ABAC rule language [4]. The similarities and differences in required functionality can be seen below:

1) *Rule*: ABAC rules are themselves models that specify their own metadata and a subset of attributes required for the rule evaluation. Unlike schemas, all of the attributes specified are not required to make the access request valid.

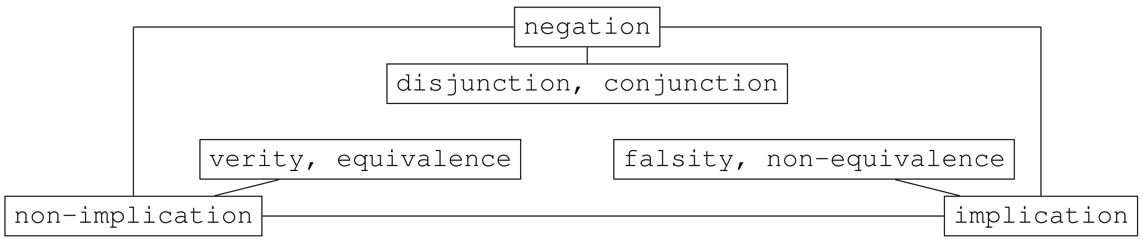

Fig. 2. Wernick's Functionally Complete Two-Function Sets [15]

TABLE I
COMPLETE SET OF INPUT AND OUTPUT VALUES FOR BINARY LOGIC [15]

p	q	H_{kpq}	a	b	c	d	e	f	g	h	a'	b'	c'	d'	e'	f'	g'	h'
1	1	α_{k11}	0	1	0	0	0	1	1	1	1	0	1	1	1	0	0	0
1	0	α_{k10}	0	0	1	0	0	1	0	0	1	1	0	1	1	0	1	1
0	1	α_{k01}	0	0	0	1	0	0	1	0	1	1	1	0	1	1	0	1
0	0	α_{k00}	0	0	0	0	1	0	0	1	1	1	1	1	0	1	1	0

2) *Attribute Interaction*: ABAC rules use attributes as the data they interact with, functionally the required functionality is equivalent.
3) *Evaluation*: Rule evaluation follows the same process as validating data against a schema. The key difference is failing the evaluation — for whatever reason — does not cause an error as it would in schema validation, instead it means that the access request must be denied. The difference between schema validation and rule evaluation is in the interpretation of the result.
4) *Serialization*: As ABAC rules are themselves models, they too require the ability to be serialized, for the same reasons. This ability is inherited from the modeling language which further supports the claim that modeling languages are a logical starting point for an ABAC rule language.

Modeling languages that do not conform to these requirements would not be suitable for use in an ABAC rule language.

B. Functional Completeness

Wernick defines functional completeness [15] as follows:

Functionally Complete 1: Given a set of truth values $B = \{0, 1\}$, variables "p" and "q" and a result α such that $p, q, \alpha \in B$, there exists 16 functions such that

$$H = \{a, b, ..., h, a', b', ..., h'\} \tag{1}$$

where

$$a'(p,q) = \neg a(p,q), \quad b'(p,q) = \neg b(p,q), \quad ... \tag{2}$$

and

$$H_k(p,q) = \alpha_k \tag{3}$$

such that they produce the output for α given specific input values for p, q as seen in Table I.

Functionally Complete 2: A set of complete functions (Ω) is defined as the set of functions such that there exists two or more functions F such that

$$F(\Omega_i(p,q), \Omega_j(p,q)) \tag{4}$$
$$= H_k(p,q) = \alpha_k,\ \Omega \subset H,\ \exists i,\ \exists j,\ \exists F,\ \forall k$$

for each value of k to produce the same output α per input p, q as seen in Table I where H is determined by equations 1, 2 and 3. Such a set of functions is said to be functionally complete.

Functional completeness is a requirement for ABAC rules to account for any possible combination of attributes with all available boolean logic. It is therefore a requirement of the validation functionality of the base modeling language to be functionally complete to be eligible for use in an ABAC rule language.

In 1942, Wernick published a proof of functional completeness [15]; in his proof, he shows that specific two-function sets are functionally complete. Figure 2 shows a diagram of function combinations that yield functionally complete two-function sets which can be read as follows:

> *Any function in one box, may be paired with a function in another box to which it is joined by a line, to form a functionally complete two-function set.*

Thus, according to the diagram in Figure 2, functional completeness can be asserted by the following two-function sets:

- negation, non-implication
- negation, implication
- negation, disjunction (`OR`)
- negation, conjunction (`AND`)
- non-implication, verity ($a'(p,q) = p \vee \neg p = 1$)
- non-implication, equivalence (`EQ`)
- implication, falsity ($a(p,q) = p \wedge \neg p = 0$)
- implication, non-equivalence (`NEQ`)

IV. JSON Schema

JSON Schema is a draft modeling language — v4 at the time of writing — for the JSON serialized data format [7] and provides similar functionality that XML Schema does for XML. This research proposes JSON Schema as a language for use in an ABAC rule language and demonstrates its viability in comparison to XML [1] and XML Schema [2] which are used in XACML [4].

A. Modeling Language Compatibility

As discussed in Section III-A, modeling languages provide a logical starting point for an ABAC rule language due to the common functionality between the two. Listing 1 shows an example of JSON Schema, without any additional functionality, being used as a basic ABAC rule.

```
{
  type: 'object',
  properties: {
    subject: {
      type: 'object',
      properties: {
        active_project: {
          type: 'string',
          enum: ['Top Secret']
        }
      },
      required: ['active_project']
    }
  },
  required: ['subject']
}
// schema.validate({
//   subject: {
//     active_project: 'Top Secret'
//   }
// }) => { validation: true, errors: []}
```

Listing 1. Example JSON Schema Rule

The rule can be read as follows:

Allow access if the subject's active project is called "Top Secret".

In Section III-A certain conditions were synthesized for modeling languages to be useful as a base syntax for an ABAC rule language. JSON Schema cannot be considered a viable candidate for use in a ABAC rule language for the JSON object model paradigm if it does not meet those criteria. The criteria and how JSON Schema satisfies each one are discussed below:

1) *Model/Rule*: JSON Schema defines JSON object models' form and metadata in a similar way to XML Schema does for XML. Both JSON Schema and XML Schema provide a standard definition of their syntax and allow models defined to be indefinitely nested with subschemas. This nesting is what defines the form of the data. The metadata for each field is determined through the use of a variety of keywords on the leaf nodes. Both languages also allow for subschemas to be references to other schema documents by Universal Resource Identifier (URI). Additionally, internal references may be made with `ref` elements and JSON Pointers [16] for XML Schema and JSON Schema respectively.
2) *Data/Attribute Interaction*: Both JSON Schema and XML Schema provide object models that define named fields, constraints and other metadata thereof. These fields are the leaf nodes in schema documents. In the validation process the name of a field in the data can be used to determine the constraints and metadata of that field from the model.
3) *Validation/Evaluation*: Both JSON Schema and XML Schema have validation keywords which are specified on each field (as required) as model metadata to define constraints on the data that can be contained in that field. These constraints are what allow ABAC rules to be arbitrarily complex and therefore must be functionally complete; for JSON Schema this is discussed ahead in Section IV-B. Most mature programming languages have several competing JSON Schema and XML Schema validation libraries that implement the functionality defined in the relevant specifications.
4) *Serialization*: JSON Schema is based on JSON and therefore inherits its serializable property. JSON is represented in plain text. Its syntax is sufficient to serialize and deserialize any JSON document for transmission and storage without any semantic loss.

B. Functional Completeness of JSON Schema

As discussed in Section III-B, one of the requirements for ABAC rules is that the language used is functionally complete. JSON Schema is functionally complete if it provides either logical `OR` or `AND` and logical `NOT` operations. JSON Schema Validation [8] provides the following relevant interfaces to that end:

- `anyOf`: The `anyOf` keyword in JSON Schema is equivalent to logical `OR`. This keyword accepts an array of schema objects and returns a successful validation if the value provided matches at least one schema in the given array.
- `allOf`: The `allOf` keyword in JSON Schema is equivalent to logical `AND`. This keyword accepts an array of schema objects and returns a successful validation if the value provided matches all of the schemas in the given array.
- `oneOf`: The `oneOf` keyword in JSON Schema is equivalent to logical `XOR`. This keyword accepts an array of schema objects and returns a successful validation if the value provided matches *only* one schema in the given array.
- `not`: The `not` keyword in JSON Schema is equivalent to a logical `NOT`. This keyword accepts a schema object and returns a successful validation if the value provided does not match the given schema object. The given schema object may contain any of the JSON Schema Validation keywords including those defined above and therefore can be used to arbitrarily combine boolean logic.

Thus, according to the proof of functional completeness by Wernick [15], JSON Schema is proven to be functionally complete as it provides both logical `OR` and `AND` and logical

`NOT` operations which can be arbitrarily combined.

JSON Schema meets the modeling requirements discussed in Section III-A, is functionally complete and provides similar functionality to the JSON object model paradigm that XML Schema does for XML. Given the alignment in functionality between the two and the fact that XML Schema has been successfully used in XACML, JSON Schema is therefore a viable candidate for use in the creation of an ABAC rule language for the JSON object model paradigm.

V. Conclusion

This research proposes JSON Schema as a modeling language from which to base an ABAC rule language. In order for such a proposal to be possible, JSON Schema must provide the necessary functionality in order to be viable for the task. The conditions of viability are discussed in Section III and the viability of JSON Schema for such a task is discussed in Section IV.

A. Viability of JSON Schema

JSON Schema is shown to fulfill all of the conditions for viability as a base modeling language for an ABAC rule language, including functional completeness which is imperative for the arbitrarily complex access control rules of ABAC.

JSON Schema was designed primarily for the JSON object model paradigm and the use thereof is therefore superior to the JSON Profile for XACML which, while it is a JSON specification, caters to the XML object model paradigm.

While — functionally — JSON Schema and XML Schema alone are capable of being used to enact ABAC in their respective paradigms, ABAC in the XML object model paradigm has benefited from the existence of the XACML standard. Given the mirrored functionality between XML Schema and JSON Schema for the different paradigms, it is possible to replace XML Schema with JSON schema in the creation of a new ABAC standard. The new standard would take advantage of the same benefits that XML Schema provides for XACML.

B. Result

JSON Schema can be safely proposed as a base modeling language for an ABAC rule language for the JSON object model paradigm as it has been shown to be viable according to the functionality required.

C. Future Work

Future work to extend this research will be targeted at the creation and standardisation of ABAC for the JSON object model paradigm. Although subjective, the wide use of the XACML standard for the XML object model paradigm is indicative that it provides a benefit to implementers and end-users alike over the basic ABAC functionality that could be provided by XML Schema alone. Particular benefits and functionality that will be targeted in a new ABAC standards are the following:

- Abstraction of a JSON-based ABAC rule language through the use of JSON Schema and JSON Schema Validation meta-schemas.
- Specification of inter-system communication to support standardised interoperability.

A new ABAC syntax will provide a simpler interface for writing ABAC security policy and detract from the verbosity of JSON Schema by providing defaults for JSON Schema validation rules for use in ABAC. This would additionally be complimented at a later stage by a policy creation tool that would automate the writing of the rule set itself by allowing end-users to specify the conditions of the policy through a GUI.

The specification of inter-system communication as it is implemented in XACML provides a number bespoke subsystems that can be individually clustered for increased performance where it is needed most. Additionally such a specification allows for different implementations of such a system to seamlessly interact. A new standard would similarly benefit from this.

Additionally, the combined use of ABAC and verified attributes[1] could potentially improve network performance through a significant decrease in traffic arising from authorization requests.

Acknowledgement

The authors would like to thank Prof. Alex "Sandy" Pentland, Thomas Hardjono and MIT Connection Science for their input and support of this research.

This work was undertaken in the Distributed Multimedia CoE at Rhodes University, with financial support from Telkom SA, Tellabs/CORIANT, Easttel, Bright Ideas 39, THRIP and NRF SA (UID 90243). The authors acknowledge that opinions, findings and conclusions or recommendations expressed here are those of the author(s) and that none of the above mentioned sponsors accept liability whatsoever in this regard.

References

[1] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau, "Extensible Markup Language (XML) 1.0 (Fifth Edition)," *W3C Standard*, 2008, Available at: https://www.w3.org/TR/xml/. [Online]. Available: https://www.w3.org/TR/2008/REC-xml-20081126/

[2] S. Gao, C. M. Sperberg-McQueen, and H. S. Thompson, "W3C XML Schema Definition Language (XSD) 1.1 Parts 1 and 2," *W3C Standard*, 2012, Available at: https://www.w3.org/TR/xmlschema11-1/ and https://www.w3.org/TR/xmlschema11-2/. [Online]. Available: https://www.w3.org/TR/xmlschema11-1/

[3] B. Parducci and H. Lockhart, "XACML Data Loss Prevention / Network Access Control (DLP/NAC) Profile Version 1.0," *OASIS Standard*, February 2015, Available at: http://http://docs.oasis-open.org/xacml/xacml-3.0-dlp-nac/v1.0/cs01/xacml-3.0-dlp-nac-v1.0-cs01.html. [Online]. Available: http://http://docs.oasis-open.org/xacml/xacml-3.0-dlp-nac/v1.0/cs01/xacml-3.0-dlp-nac-v1.0-cs01.html

[1] Verified attributes are digital artifacts that contain data; the verify of which can be verified independently of its most recent origin.

[4] E. Rissanen, "eXtensible Access Control Markup Language (XACML) Version 3.0," *OASIS Standard*, January 2013, Available at: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-cd-1-en.html

[5] D. Brossard, "JSON Profile of XACML 3.0 Version 1.0," *OASIS Standard*, vol. 201401, 2014.

[6] M. Nottingham. (2012, April) JSON or XML: Just Decide. Personal Blog. Accessed: 2016-10-30. Available at: https://www.mnot.net/blog/2012/04/13/json_or_xml_just_decide. [Online]. Available: https://www.mnot.net/blog/2012/04/13/json_or_xml_just_decide

[7] A. Wright, "JSON Schema: A Media Type for Describing JSON Documents," *IETF Standard*, 2016, Internet Draft v4. [Online]. Available: http://json-schema.org/latest/json-schema-core.html

[8] A. Wright and G. Luff, "JSON Schema Validation: A Vocabulary for Structural Validation of JSON," *IETF Standard*, 2016, Internet Draft v4. [Online]. Available: http://json-schema.org/latest/json-schema-validation.html

[9] R. Richardson, "CSI computer crime and security survey," *Computer Security Institute*, vol. 1, pp. 1–30, 2008.

[10] R. S. Sandhu and P. Samarati, "Access control: principle and practice," *Communications Magazine, IEEE*, vol. 32, no. 9, pp. 40–48, 1994.

[11] K. Kent and M. Souppaya, *Guide to computer security log management: recommendations of the National Institute of Standards and Technology (NIST).* US Department of Commerce, Technology Administration, National Institute of Standards and Technology, 2006.

[12] E. Yuan and J. Tong, "Attributed based access control (ABAC) for web services," in *Proceedings of IEEE International Conference on Web Services and ICWS, 2005.* IEEE, 2005.

[13] T. Priebe, W. Dobmeier, C. Schläger, and N. Kamprath, "Supporting attribute-based access control in authorization and authentication infrastructures with ontologies," *Journal of Software*, vol. 2, no. 1, pp. 27–38, 2007.

[14] V. C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to Attribute Based Access Control (ABAC) Definition and Considerations," *NIST Special Publication*, vol. 800, p. 162, 2014.

[15] W. Wernick, "Complete sets of logical functions," *Transactions of the American Mathematical Society*, vol. 51, no. 1, pp. 117–132, 1942. [Online]. Available: http://www.ams.org/journals/tran/1942-051-00/S0002-9947-1942-0005281-2/S0002-9947-1942-0005281-2.pdf

[16] P. Bryan and K. Zyp, "JavaScript Object Notation (JSON) Pointer Specification," *IETF Standard*, 2011.

Gregory Linklater is a postgraduate student at Rhodes University. His research interests include distributed and user-centric identity, verified attributes and anonymous authorization. He is currently completing his Masters degree in computer science with the help and support of MIT CONNECTION SCIENCE.

Christian Smith is the lead software engineer of OPAL at MIT CONNECTION SCIENCE.