AQUA: An Efficient Solver for the User Authorization Query Problem

Alessandro Armando
alessandro.armando@unige.it
DIBRIS, Università di Genova
Genova, Italia

Giorgia A. Gazzarata
giorgia.gazzarata@unige.it
DIBRIS, Università di Genova
Genova, Italia

Fatih Turkmen
f.turkmen@rug.nl
University of Groningen
Groningen, Netherlands

ABSTRACT

We present AQUA, a solver for the User Authorization Query (UAQ) problem in Role-Based Access Control (RBAC). The UAQ problem amounts to determining a set of roles granting a given set of permissions, satisfying a collection of authorisation constraints (most notably Dynamic Mutually-Exclusive Roles, DMER) and achieving some optimization objective, i.e. seeking min/max/any number of roles to activate and/or permissions to grant. AQUA supports the enforcement of a wide class of DMER constraints as well as several types of optimization objectives (namely, min/max/any number of roles to activate, min/max/any number of permissions to grant, and a combinations thereof). In this paper, we demonstrate the use of AQUA over a running example while providing certain implementation details including the architecture.

ACM Reference Format:

Alessandro Armando, Giorgia A. Gazzarata, and Fatih Turkmen. 2020. AQUA: An Efficient Solver for the User Authorization Query Problem. In *Proceedings of SACMAT'20: ACM Symposium on Access Control Models and Technologies (SACMAT) (SACMAT '20)*. ACM, New York, NY, USA, 3 pages. <https://doi.org/10.1145/nnnnnnn.nnnnnnn>

1 INTRODUCTION

Role-based access control (RBAC) has proven to be one of the most successful access control models thanks to its intuitiveness, simplicity and standard support. Today, many authorization systems support at least one form of RBAC among several variants the standard [11] provides. One of the key issues in RBAC is, *given a set of requested permissions, finding an optimal set of roles in a given session that will both cover the requested permissions and satisfy a collection of Dynamic Mutually-Exclusive Roles (DMER) constraints*. This problem, known as the User Authorization Query (UAQ), has been well studied in the literature, however there is little tooling (e.g. solver, benchmark generator) support to study its different aspects.

In this paper we present AQUA, a solver for the UAQ problem that combines (i) a reduction of various types of DMER constraints

(in a given state) to an equivalent set of single-session DMER constraints, (ii) a reduction of the UAQ Problem to the Weighted Maximum Satisfiability problem (Weighted MAX-SAT), and (iii) the application of state-of-the-art Weighted MAX-SAT solvers to tackle the problem. To the best of our knowledge AQUA is the only solver that simultaneously offers all the above features.

1.1 Running Example

Consider a data visualization application displaying plots about a pandemic and RBAC is employed to control access to (sensitive) plots based on user's privileges. Table 1 summarizes the RBAC policy in terms of PA relation. In the RBAC policy, there is a single operation, "view", that can be performed on the plots and thus the plots uniquely identify the permissions. We assume that there is a single user (*alice*) and a single session (s_1) in the system for the simplicity of discussion.

Roles	Permissions
Employee	$\{p_1, p_2\}$
Guest	$\{p_1, p_2\}$
Researcher	$\{p_3, p_4, p_5\}$
BoardMember	$\{p_3, p_4\}$

Table 1: PA for the Visualization Application

The system has a single DMER constraint;

$$DMER(\{Researcher, BoardMember\}, 2)$$

that makes sure that a user cannot assume the permissions of a researcher and a board member at the same time.

Besides some required plots, $\{p_1, p_2\}$, accessible to some basic roles (i.e. Employee, Guest) the application **must** also display a maximal set of plots, $\{p_3, p_4, p_5\}$, accessible only to more privileged roles (i.e. *Researcher*, *BoardMember*). The intuition is the application should maximize the amount of information conveyed to the user, i.e. provide a coherent overview from different sources and combine them in a meaningful way, while complying with the authorization constraints.

This amounts to the following UAQ query:

$$q = (s_1, \underbrace{\{p_1, p_2\}}_{P_{lb}}, \underbrace{\{p_1, p_2, p_3, p_4, p_5\}}_{P_{ub}}, \underbrace{\max}_{obj})$$

Given the *max* objective, the query prioritizes *availability* of the permissions over the *safety* of the system.

2 IMPLEMENTATION DETAILS

AQUA has been mostly implemented in C with various functionalities written in Java or implemented as shell scripts. The implementation of core architectural components (see Figure 1) constitutes of

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.

SACMAT '20, June 10–12, 2020, Barcelona, Spain

© 2020 Association for Computing Machinery.

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...\$15.00

<https://doi.org/10.1145/nnnnnnn.nnnnnnn>

some ~2K lines of code. The source code of AQUA is freely available¹ as part of a larger project presented in [3] and [4] along with further details on the performance. In what follows, we provide details about the implementation and demonstrate the use of AQUA over the running example.

2.1 Architecture

AQUA receives a UAQ instance and returns a set of roles. Figure 1 presents the overall architecture. There are four components:

- **Format Translator:** AQUA has a native input format for UAQ specifications, however it can also work with different formats. The component *Format Translator* is responsible for converting input specifications written in different formats to AQUA’s native format. The current implementation supports the notation used in [10].
- **Instance Parser:** The component *Instance Parser* parses the input specifications and generates an internal representation for the query and the accompanying RBAC model.
- **MaxSAT Encoder:** Given the internal representation, the component *MaxSAT Encoder* generates the corresponding MaxSAT formulae in WCNF format [5]. Among others, these formulae include clauses for RBAC relations (*UA*, *PA*), optimization objective and DMER constraints. Note that, AQUA employs the inequality encoding presented in [12] for efficiently handling DMER constraints, however it can also be used without this encoding.
- **Solver Interface:** AQUA can work with different MaxSAT solvers to tackle the final MaxSAT instance and the component *Solver Interface* enables the easy integration of MaxSAT solvers. This component is mainly for providing syntactical adjustments to the inputs sent to the solver, invokes the solver with the input and extracts the relevant information from its console output.

Figure 1: AQUA Architecture

2.2 Execution

Before running AQUA, a few conditions need to be satisfied apriori:

- The selected MaxSAT solvers (AQUA supports various solvers natively including Loandra [6] and [1]) needs to be installed and the relevant environment variable needs to be set.
- If the cardinality encoding from [12] is used then the relevant executables need to be installed.

After these requirements are satisfied, AQUA can be invoked against a given UAQ specification (denoted as *UAQIns*) as follows:

¹<https://github.com/alessandroarmando/aqua>

```
$ ./solve_uhq_smart.exe -s $MaxSATSolver $UAQIns
```

Assuming that the solver *Loandra* has been chosen, AQUA will show the roles (Figure 2) to be activated along with some additional information from the MaxSAT solver. It can also provide certain debug information related to errors in the input specification or misconfigurations regarding the MaxSAT solver integration.

```

1  satsolver = Loandra
2  ...
3  Parsing /source/inst_examples/DemoUAQ.res
4  Reading model ...
5  SOLUTION: -1 2 3 -4 5 6 7 8 9 -10 -11 -12 -13 -14 -15 -16 -17 18 -19
6  ANSWER: -Employee Guest Researcher -BoardMember p1 p2 p3 p4 p5

```

Figure 2: Output from AQUA

As can be noted from output, the solution obtained from the solver (Line 5) and the permissions associated with the selected roles are also printed (Line 6).

3 CONCLUSIONS

We presented AQUA, a solver for the UAQ problem that implements a reduction of the UAQ Problem to Weighted MAX-SAT and can thus leverage state-of-the-art Weighted MaxSAT solvers taken off-the-shelf to efficiently solve UAQ problems. AQUA supports the enforcement of a wide class of DMER constraints and several types of optimization objectives. It is freely available and can be further extended with new optimization strategies and objectives. Moreover, thanks to the flexible architecture, new MaxSAT solvers can be easily integrated for efficiency purposes and/or better features.

REFERENCES

- [1] Mario Alviano, Carmine Dodaro, and Francesco Ricca. 2015. A MaxSAT Algorithm Using Cardinality Constraints of Bounded Size. In *Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015*. 2677–2683.
- [2] Carlos Ansótegui. 2012. WPM1-12. https://ulog.udl.cat/?page_id=30
- [3] Alessandro Armando, Giordina Gazzarata, and Fatih Turkmen. 2020. Benchmarking UAQ Solvers. In *To Appear in Symposium on Access Control Models and Technologies, SACMAT 2020, Barcelona, Spain*.
- [4] Alessandro Armando, Silvio Ranise, Fatih Turkmen, and Bruno Crispo. 2012. Efficient run-time solving of RBAC user authorization queries: pushing the envelope. In *Second ACM Conference on Data and Application Security and Privacy (CODASPY)*. 241–248.
- [5] Fahiem Bacchus, Matti Järvisalo, and Ruben Martins. 2019. MaxSAT Evaluation 2018: New Developments and Detailed Results. *J. Satisf. Boolean Model. Comput.* 11, 1 (2019), 99–131.
- [6] Jeremias Berg, Emir Demirovic, and Peter J. Stuckey. 2019. Core-Boosted Linear Search for Incomplete MaxSAT. In *Integration of Constraint Programming, Artificial Intelligence, and Operations Research - 16th International Conference, CPAIOR 2019, Thessaloniki, Greece, June 4-7, 2019, Proceedings*. 39–56.
- [7] Daniel Le Berre and Anne Parrain. 2010. The Sat4j library, release 2.2. *JSAT* 7, 2-3 (2010), 59–6.
- [8] Federico Heras and Javier Larrosa. 2006. New Inference Rules for Efficient MaxSAT Solving. In *Proceedings, The Twenty-First National Conference on Artificial Intelligence and the Eighteenth Innovative Applications of Artificial Intelligence Conference, July 16-20, 2006, Boston, Massachusetts, USA*. 68–73.
- [9] M. Koshimura. 2014. QMaxSAT: Q-dai MaxSAT Solver. In <http://sites.google.com/site/qmaxsat/>.
- [10] Nima Mousavi and Mahesh V. Tripunitara. 2012. Mitigating the Intractability of the User Authorization Query Problem in Role-Based Access Control (RBAC). In *NSS*. 516–529.
- [11] National Institute of Standards and Technology (NIST). 2004. Role-Based Access Control. *American National Standards Institute, Inc.* (2004).
- [12] C. Sinz. 2005. Towards an Optimal CNF Encoding of Boolean Cardinality Constraints. In *Principles and Practice of Constraint Programming (CP)*. 827–831.

A MAXSAT INTEGRATION

One of the strengths of AQUA is it employs off-the-shelf MaxSAT solvers to tackle the respective UAQ encodings. Since the MaxSAT research has a lively community, new advances in performance (and the features) will positively impact the capabilities of AQUA as a side product. The current version of AQUA promptly supports a range of MaxSAT solvers including: qmaxsat [9], Sat4j [7], MaxSATz [8], WPM1 [2], maxino [1] and Loandra [6]. New MaxSAT solvers can be easily integrated thanks to the standard format specification provided by the MaxSAT competition [5] in input and output formats.

B INPUT FORMAT

AQUA has a native syntax to encode UAQ instances. The syntax allows the specification of all relevant RBAC details (policy relations, sessions, history, constraints) in a concise form and it is intuitive. Figure 3 presents the encoding of the running example in this syntax.

```
1  users : alice ;
2  roles : Employee Guest Researcher BoardMember ;
3  perms : p1 p2 p3 p4 p5 ;
4  sesss : s1 ;
5
6  sof [ s1 ] : alice ;
7  --
8  ua [ alice ] : Employee Guest Researcher BoardMember ;
9  --
10 pa [ Employee ] : p1 p2 ;
11 pa [ Guest ] : p1 p2 ;
12 pa [ Researcher ] : p3 p4 p5 ;
13 pa [ BoardMember ] : p3 p4 ;
14 --
15 yesterday [ s1 ] : ;
16 --
17 once [ s1 ] : ;
18 --
19 mer ss d 2 Researcher BoardMember ;
20 --
21 QUERY s1 MAX GRANT p3 p4 p5 DENY ;
```

Figure 3: Input Format for UAQ Instances

Note that there are additional constructs such as *yesterday* (line 15) and *once* (line 17) in the input syntax to support the specification of history-based enforcement of MER constraints. The MER constraint of the example is specified at Line 19 which states that the constraint is enforced per session (i.e. does not span several sessions) and prevents simultaneous activation of the roles *Researcher* and *BoardMember* by the user. Finally, the Line 21 encodes the UAQ that requires the maximum set of permissions from $P_{ub} \setminus P_{lb} = \{p_3, p_4, p_5\}$ while achieving all permissions in P_{lb} .