Data flow analysis from capability lists, with application to RBAC and LaBAC

Abdelouadoud Stambouli and Luigi Logrippo

Université du Québec en Outaouais
Department of Computer Science and Engineering
Gatineau, Québec, Canada
J8Y 3G5
staa16@uqo.ca, luigi@uqo.ca

Abstract. For the analysis of access control networks, where capability lists, access control matrices, or RBAC permissions are available, it can be very useful to be able to determine which subjects can be able to know, or which objects can be able to store, data originating from objects in the network. This information can be used in order to answer questions of secrecy, integrity and privacy, related to the data flow analysis problem. On the basis of a logical method, we present a graphical formalism capable to represent such networks and for which the data flow problems can be defined. We present algorithms to calculate answers to data flow questions. Complexity analysis and simulations show that these questions can be practically answered for networks of sizes up to several tens of thousands of subjects and objects, which is the size of many real-life organizations. We also show that the results obtained can be used in the process of role engineering in Role based access control, for determining secrecy levels, as well as for eliminating or combining roles or objects. Finally, a method is demonstrated to go from capability lists to Label-based or Multi-level access control systems.

Keywords: Access control, Data flow, Data security, Data secrecy, Algorithms, Role based access control, Role engineering, Label-based access control, Multi-level access control.

1 Introduction and motivation

A fundamental question in research on information security and privacy in data networks is the following: assuming that certain data are available somewhere in a network, where else can they end up? To answer this question realistically, one must consider the information supports and the information paths available in the network. Obviously, in general this is an intractable problem, even difficult to define, because information can move, or can be blocked, in a network in many different ways, depending on access control, encryption, inference, channel characteristics, covert channels, protocols, errors, human intervention, etc.

We take a view based on access control methods according to established access control theory. First, we consider data only and not information (namely, we do not take into account data flows generated by inferring information from data and then encoding it in other data). Second, we assume that data can only be available in a network by being *known to subjects* or *stored in objects*. Third, we assume that data can move between subjects and objects over perfect channels according to policies specified by access control matrices, capability lists, permission in Role based access control, routing tables, or other methods directly translatable into any of these. If an object can store some data, and a subject can read from the object, then the subject can know the data. The subject can then write these data in other objects where they will be stored and from where other subjects can read them and so on. We show that, in a snapshot of the system where the capabilities do not change, the problem is very tractable and allows solutions that can be stepping-stones towards other more complex problems.

Essentially, our approach is based on the following steps:

1. Using capability lists, we construct logical models, represented as graphs, of all possible data transfer paths in networks; the nodes in the graphs are subjects or objects, and the edges are reading and writing capabilities.
2. These models can then be explored in order to see where the data contained in certain data objects can possibly end up (the data flow problem).

Extended version of paper titled: Data flow analysis from capability lists, with application to RBAC. Information Processing Letters (Elsevier) 141 (2019) 30–40

As a byproduct of these steps, we are also able to determine secrecy areas and levels in the network. For a given object, we can determine that it is a *secret of* certain subjects and objects. For a network, we can determine what are its less or more secret subject and objects.

Our method is shown to be practically feasible up to a fairly large number of subjects and objects, enough to be able to apply it to many realistic systems. Although the goal is easily seen, practically important and not difficult to achieve, we are not aware of similar experiences having been reported in the literature, or of industrial tools having been produced towards it – two research tools are reported in [1] [10] (see below), but without time estimates. Therefore, we believe that this paper fills a void, and will inspire research towards applications, improvements and extensions.

Being able to practically solve our problem is important in order to answer several questions in security and privacy:

- The *secrecy* question: can secret data be known or stored where they shouldn't?
- The *integrity* question: can data coming from unreliable databases end up being stored in data bases that should be highly reliable?
- The *availability* question: can data reach users or data bases where it might be needed?

We briefly consider the application of our method to RBAC [7][19]. In spite of the practical success of RBAC, few publications address the problem of data flow analysis in RBAC, although this problem is easily understood. We present certain conclusions that can be reached by the analysis of such flows, which have consequences for the placement of secret data in RBAC systems and for role engineering. Concerning this second point, we show that certain roles can be considered unnecessary, either because they cannot receive any data, or because they can be merged with others.

The paper is structured as follows: Section 2 is our concise literature review. In section 3 we review some basic concepts and present our basic definitions. Section 4 presents a small example to illustrate our goals and methods. Section 5 presents the algorithms we use, with an evaluation of their theoretical complexity, leading to the conclusion that we need only polynomial algorithms. Section 6 presents our simulation method and simulation results. Section 7 discusses a specific technical point concerning the two equivalence relationships that we use in our work. Section 8 provides a larger example. Section 9 discusses some applications of our method to RBAC, especially for role engineering. In Section 10 a method is presented to go from capability lists to Label-based access control systems. Section 11 concludes the paper and mentions subjects for future research, which include methods for flow control in the Cloud and in the Internet of Things.

This paper is dedicated to the memory of Dr. Sylvia Louise Osborn, a pioneer in this research area and a source of encouragement for our work.

2 Literature review

While the literature on data flow control is extensive, there is almost no literature on algorithms and simulation results for data flow analysis in access control networks. We will mention only a few papers that we consider specifically related to our work.

Several of these papers are related to RBAC. The basic paper on data flow analysis in RBAC is the one of Osborn [20] in which it is shown how a role graph can be mapped on a data flow graph. The resulting data flow graph is proven to be the result of a flow relationship between objects, this relationship is derived from the permission assignments in consideration of role relationships, and constraints on sessions between different roles.

Gofman et al. [10] proposed an optimization of Osborn's algorithm using incremental methods, the resulting algorithm is proven to be less time and space consuming. Their algorithm provides also information such as the nature of the data flow, whether it is direct or if it is by transitivity. These points will be developed in our paper. A follow up of this article [11] presented a tool RBAC-PAT for the analysis of properties of RBAC and ARBAC policies including information flow.

Amthor et al. [1] present WorSE, a comprehensive workbench for model-based security engineering. They use both a state-based representation of security systems, and a relational one such as ours. Similar to us, they aim to detect indirect flows, which they call *covert flows*. They also use the concept of information flow graphs, which they reduce according to information equivalence relationships.

The last three papers will be further discussed in our Conclusions.

Several papers have dealt with methods for blocking illegal flows in RBAC, or information flow control. Samarati et al. [21] proposed a model that overcomes the vulnerability of discretionary access control by completing it with mechanisms to prevent illegal flows caused by Trojan horses. Izaki et al. [13] use RBAC to control information flow. They define safe roles where only legal flows occur, then objects methods are classified and used to construct an information flow graph in which illegal flows can be identified. Nakamura et al. [19] developed a synchronization protocol to prevent illegal flows in RBAC systems, where operations are blocked if they illegally write or read over objects. The same direction is taken by Chon and al. in [5] but this time a role lock over objects is introduced, the objects are locked before any operation and an operation is aborted if the locking fails to prevent illegal flow. Zhang and Yang [24] clearly explain the problem of data leakage in RBAC-based models and propose the use of Mandatory access control methods to produce label assignments to make leakage impossible. Although some of these approaches are related to ours, the aim of our research is of determining efficiently what data flows are possible, given certain access control capabilities, rather than of controlling the flow of information, or of blocking certain flows.

In the abundant literature on RBAC role mining and engineering there is no mention of the possibility of merging or eliminating roles because of data flow characteristics, as it will be discussed in Section 8.

We have reviewed industrial documents describing RBAC audits, and we have found that these limit themselves to questions of appropriate role assignment, role abuse, role conflicts, etc.

In any case, our methods apply well beyond RBAC, to all systems for which reading or writing authorizations among subjects and objects can be established, see Section 3 and the Conclusions.

3 Basic concepts

We consider two types of *entities*: *subjects* and *objects*. The lower case letter *s*, with various primes or subscripts, will be used for variables for subjects, while the upper case letter *S* with various numerals will be used for constants over subjects. Similarly, the lower-case letter *o* will be used for variables over objects, while the upper case letter *O* will be used for constants over objects. The letter *e* will be used for variables of entities, when we will introduce concepts that hold whether an entity can be a subject or an object.

We start by reviewing and adapting to our needs some basic definitions first proposed by Lampson [15], and repeated many times in the literature, in many variations. In a system with *i* subject and *j* objects, an *access control matrix* is an $i \times j$ matrix *M* where the element $M_{m,n}$ is a set of capabilities, telling what are the capabilities of subject s_m over object o_n. A row *m* of the matrix is the list (or set) of capabilities of s_m. We consider only the capabilities of *reading* and *writing*. We choose to present capability lists as sets of relations. Following [16] we use the constant predicates *CR* and *CW* (for *CanRead* and *CanWrite* respectively) :

- *CR(s,o)* is true if subject *s* has read capability on *o*.
- *CW(s,o)* is true if subjects *s* has write capability on *o*.

So capability lists or access control matrices can be represented as sets of *CR* and *CW* relationships. An example is given in Table 2. By using the predicates just presented, Table 2 can be described thus: {*CW(S1,O3), CR(S2,O1), CR(S2,O2), CR(S2,O3), …}*. Intuitively, the existence of a *CR* (or *CW*) relationship between a subject and an object means that the subject can receive data from the object (or the subject can send data to the object). More abstractly, *CR(s,o)* can be taken to mean that any data that *o* can store, *s* can know, and similarly for *CW*. In practice, this can correspond to several situations that we may want to consider: the traditional read and write permissions as in Unix; possession of encryption-decryption keys; the existence of channels, possibly hypothetical hidden ones; network routing relationships; data transfer paths in the Internet of Things. If we take roles for subjects, and we restrict RBAC permissions to reading and writing operations on resources, RBAC permissions can be directly translated into capability lists or access control matrices and vice-versa [7 Section 3.3.2] [3].

Our simulation programs will represent access control lists in terms of Boolean arrays, as in Table1 (where *S3* can read from *O1* but cannot write in it, etc.).

Table 1. Boolean capability list for *S3* in the example of Table 2 and Fig. 1

	O1	O2	O3	O4
read	1	0	1	0
write	0	1	1	0

We also use the two additional predicates *CanKnow and CanStore* [16], respectively abbreviated *CK* and *CS* which are related between themselves and with *CR, CW* by the following axiom and inference rules:

- *The axiom: CS(o,o)* (in other words, any object can store itself).
- *The inference rule for CK is: (CR(s,o) ∧ CS(o,o')) → CK(s,o')* (i.e. if *s* can read *o*, then whatever *o* can store, *s* can know).
- *The inference rule for CS is: (CW(s,o) ∧ CK(s,o')) → CS(o,o')* (i.e. if *s* can write *o*, then whatever *s* can know, *o* can store).

We stipulate further that *CK(s,o)* or *CS(o,o')* can be true only by the inference rules or axiom.

We will also use the following auxiliary definitions:

- *CKS(s)* for a subject *s*, is the set of all objects *o* that can be known by *s*, i.e. *CKS(s) = {o: CK(s,o)}.*
- *CSS(o)* for an object *o*, is the set of all objects *o'* that can be stored by *o*, i.e. *CSS(o) = {o': CS(o,o')}.*

For a subject *s* (or object *o*), *CKS(s)* (or *CSS(o)*) is the set of all data objects that can be delivered to *s* (or stored in *o*) by sequences of read-write operations. Table 3 will show an example.

Clearly, for each *s* and *o* we can calculate these two sets from the axiom, the given *CR* and *CW* relationships, and the inference rules, and this paper introduces efficient algorithms to do this.

Our inference rules assume transitivity of the data transfer operation, i.e., that any or all data can always be passed on if a *CR* or *CW* capability exists. This is not always true in security, surely we can have entities that will pass on data according to some conditions or judgment. In our model, such situations can perhaps be represented by splitting subjects or objects according to their behavior in this respect. Transitivity is a pessimistic assumption that may lead to over-protected systems.

In [16] we used the additional concept of variable, using which the definitions are symmetric between subjects and objects. We have opted for simpler, but asymmetric, definitions in this paper by which objects can contain objects, and subjects can know objects.

For a given network, the set of subjects and objects in the network with their *CR* and *CW* relationships can be shown graphically, yielding a *network directed graph (digraph)*, by using the following conventions (see Fig.1):

- *The nodes in the digraph are the entities in the network; we use ovals for subjects and rectangles for objects.*
- *There exists a directed edge from object o to subject s iff CR(s,o)*
- *There exists a directed edge from subject s to object o iff CW(s,o).*

In choosing the notation, we have tried to strike a compromise between the usual verbal and graphical notations, which are not mutually consistent. The verbal notation puts the subject first, while the graphical notation shows the data flow. So, we represent *CR(s,o)* with an arrow from *o* to *s*, while we represent *CW(s,o)* as an arrow from *s* to *o*. Distinguishing between these two relationships is useful in access control theory, where a distinction is made between subjects and objects. But if we eliminate this distinction, and use the concept of *entity* to denote both subjects or objects, then we can define a relation *CanFlow* or *CF(e,e')*, corresponding to a directed path from *e* to *e'*.

CF(e,e') is true iff one of the following is true:

- *e=e'*
- *e' is a subject and e is an object and CR(e',e)*
- *e is a subject and e' is an object and CW(e,e')*
- *there exists e" such that CF(e,e") and CF(e",e`).*

By these definitions, *CF* is a transitive, reflexive relationship, or a *preorder*. We take the network digraph described above to represent this preorder, except for reflexive relationships that are implicit.

We also generalize the functions *CK* and *CS* by introducing the function *CanHold* or *CH*, as follows:
If *e* is a subject *s*, then *CH(e)=CK(s)*. If *e* is an object *o*, then *CH(o)=CS(o)*.

Finally, the functions *CKS* and *CSS* can be generalized into a function *CanHoldSet* or *CHS*: if *e* is a subject *s*, then *CHS(e)=CKS(s)*; if *e* is an object *o*, then *CHS(e)=CSS(o).*

A *partial order* is a relationship that is transitive, reflexive and anti-symmetric. If we partition a preorder into equivalence classes, the result is a partial order of equivalence classes: this is because the resulting relationship is transitive and reflexive, while no symmetric relationships could remain among the equivalence

classes [8 Section 2.2]. This result has its correspondent in digraph theory, as follows. A *component* in a transitive, reflexive digraph is a set of nodes that are mutually reachable, and which is maximal in the sense that it is not included in a larger set of nodes with the same property. If each component is *condensed* into a single node, then the resulting digraph represents a partial order (all the symmetric paths having been encapsulated in components). Further, this resulting digraph has the same connectivity as the original one, in the sense that if node *x* is reachable from node *y* in the original digraph, then the node that represents the component of *x* is reachable from the node that represent the component of *y* [12 Chapter 3]. For more discussion and examples on this, see [17].

We say that two entities are:

- *CF-equivalent* if they belong to the same component in their network or digraph;
- *CHS-equivalent* if they have the same CHS.

Clearly, CF-equivalence implies CHS-equivalence, but the opposite is false in very specific cases, see Section 9. This property enables us to use CF-equivalence in order to find CHS-equivalence.

4 An example

The example in this section will show the important concepts and results of our method.

Consider a network with five subjects *S1* to *S5* and four objects *O1* to *O4*. The capability lists, expressed as mentioned as *CR, CW* relationships in tabular form, are shown in Table 2, where the capabilities for *S1* are that it can only write on *O3*, and so on. Table 1 shows the same capabilities for *S3* in the form of a Boolean matrix.

Table 2. Capability lists for the network of Fig. 1

S1	CR	none
	CW	O3
S2	CR	O1,O2,O3
	CW	O2
S3	CR	O1,O3
	CW	O2,O3
S4	CR	O2,O4
	CW	O2,O4
S5	CR	O4
	CW	O4

Fig. 1 gives a graphical representation of this network, by the conventions defined in Section 3.

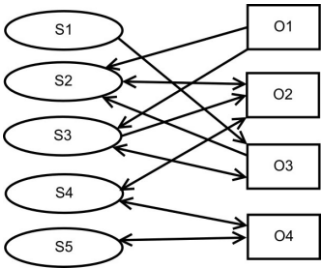


Fig. 1. Network example

A first question we can ask for such a network is: for an object *o*, what are the subjects and objects that can know or store it? We call this the *single object flow* problem. This question can be important for objects that can contain secret data. Answering this question leads to partitioning the network in two *areas*, one where the object can be known or stored (the sets of subjects and objects of which *o* is a *secret*) and another where it cannot. Fig. 2, to be read left-to-right, shows the *area of O*3 in the network of Fig. 1.

Intuitively this is clear, but formally it can be justified in terms of the inference rules in Section 3, since by Table 2 we know:

CR(S2,O3) $\wedge$ *CR(S3,O3)* $\wedge$ *CW(S2,O2)* $\wedge$ *CW(S3,O2)* $\wedge$ *CR(S4,O2)* $\wedge$ *CW(S4,O4)* $\wedge$ *CR(S5,O4)*

Using the axiom *CS(O3,O3)*, the inference rules allow us to progressively derive *CK(S2,O3)*, *CK(S3,O3)*, *CS(O2,O3)* (twice), *CK(S4,O3)*, *CS(O4,O3)*, *CK(S5,O3)*. By inspection, one can see that no other such derivations are possible for any other subjects or objects in the network.

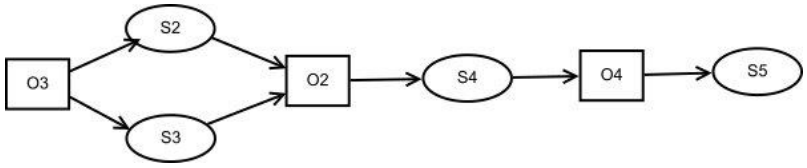


Fig. 2. The flow of object *O3* in the network, the area of *O3*

This information is useful for a system administrator wishing to audit, add or remove certain flows. As a further step, suppose that in a network there is a set of objects whose association can lead to the discovery, or inference, of some critical information. A practical example of this case could be the rule: only executives can know both tables: *Role-by-employee* and *Salary-by-role*. It is then important to determine what exactly are the entities that can know or store all such data together. If these tables are in two different objects, then to answer this question it is sufficient to repeat the above reasoning twice, once for each object.

A global answer to all questions of this type can also be given, we call this the *global object flow* problem. To do this, we calculate the *CKS* and *CSS* functions for all subjects and objects, see Table 3. E.g. for *S3*, we have seen above that *O3*$\in$*CKS(S3)*. By using the inference rules for *O1*, we will also find that *O1*$\in$*CKS(S3)*, and no other possibilities exist.

Table 3. *CanKnow* and *CanStore* sets for the network

CKS(S1)= { }	CSS(O1)= {O1}
CKS(S2)= {O1,O2,O3,O4}	CSS(O2)= {O1,O2,O3,O4}
CKS(S3)= {O1,O3}	CSS(O3)= {O1,O3}
CKS(S4)={O1,O2,O3,O4}	CSS(O4)= {O1,O2,O3,O4}
CKS(S5)={O1,O2,O3,O4}	

Looking at Table 3, we can see that:

- Several subjects (objects) can know (store) the same objects, and so they are CHS-equivalent, see for example *S3, O3*.
- There is a partial ordering of these equivalence classes, determined by the inclusion of the *CHS*, e.g. in this sense the equivalence class containing *S3,O3* is included in the equivalence class containing *S2, S4,S5,O2,O4*.

The CHS equivalences can be computed by using directly the axiom and inference rules of Section 3, as we have done above. However this would require the use of languages appropriate to program inference rules directly; such languages are interpretive and not the best in terms of computational efficiency. But since CF equivalence implies CHS equivalence, the former can be used in order to find the latter by using efficient digraph algorithms. In Section 9 we will see that there are only very limited cases where CHS equivalence does not imply CF equivalence. By looking for CF equivalences, we obtain the simplified digraph of Fig. 3, where the subjects (objects) that can know (store) an object have been grouped in the *area of* the object.

Note that Fig. 3 shows all and only the data flows of Fig. 1.

- The boxes in Fig. 3 contain entities that constitute components in the original network. They are not only CF-equivalent, but also CHS-equivalent. In other words, it can be checked that *S3,O3* and *S2,S4, S5,O2,O4* constitute components in the graph of Fig. 1 and so they must all have the same *CHS*.

- There are arrows between boxes to denote the partial order between the components. The arrows denote also *CF* relationships between the entities in the components.

This partial order allows us to arrange the network by increasing *levels of secrecy*, according to the concepts presented in [17]. On one hand, the contents of *O1* are the least secret, since they can propagate to all other subjects and objects (except for *S1*). On the other hand, it is clear that the most secret data should be stored in *O2* or *O4*: there, they can be shared with *S2*, *S4*, *S5* but they cannot propagate further.

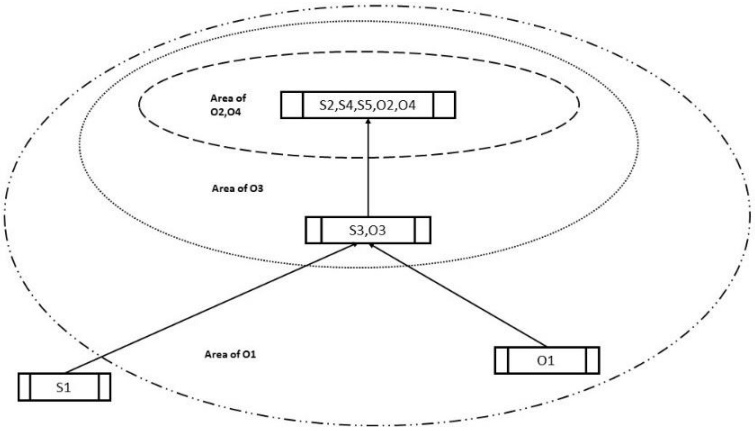


Figure 3. The partial order of equivalence classes of entities and the areas for Fig. 1

We can now construct several networks that are equivalent to the initial one (from the dataflow point of view) by rearranging subjects and objects according to the following rules:

- Within each component there can be any *CR*, *CW* relationships that generate symmetric *CF* relationships among all pairs of entities.
- Between components, there can be any edges that are consistent with the partial order: e.g. in Fig. 3 we have *CW(S3,O2)* but other possibilities would be just as good, such as some or all of: *CW(S3,O4)*, *CR(S2,O3)*, *CR(S4,O3)*, *CR(S5,O3)*, see Fig. 4.

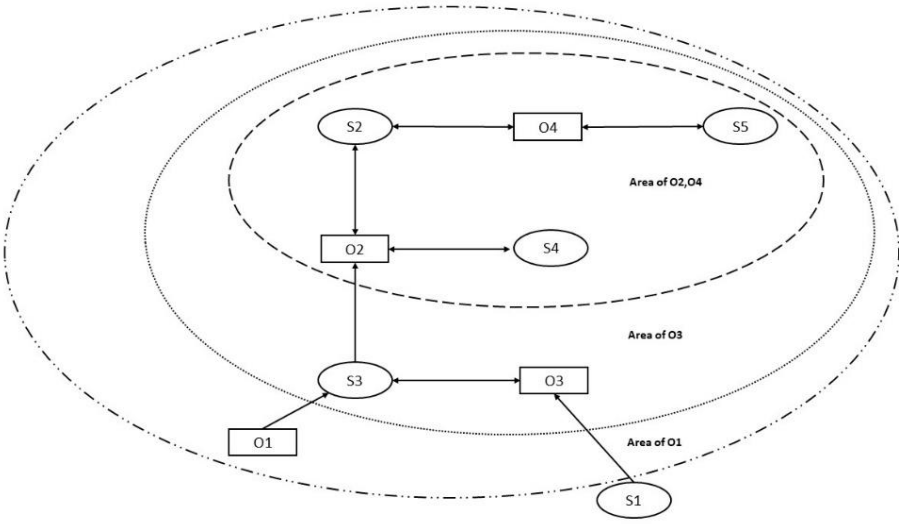


Figure 4. A network equivalent to the one of Fig. 1

The decision of performing such restructurations or simplifications normally should be left to the network administrator, because good reasons might exist for not restructuring at all, or for choosing one restructuration rather than another. For example, a network administrator may get a more efficient network by adding all the capabilities that are implied by transitivity. In general, if *CW(s,o)* is false but by transitivity there is a flow between *s* and *o*, should the administrator make *CW(s,o)* true? This is a non-obvious decision, since in practice indirect writing may depend on read and write actions (and decisions) of intermediary subjects. On the other hand, one may wish to reduce the capabilities as much as possible in order to limit the security checks.

Our algorithms can draw these diagrams but they become impractical beyond small sizes. But the algorithms can provide all the information that an administrator needs in order to do simplifications and reorganizations. Beside the *CKS*s and *CSS*s and order relationships they can provide the tables of empty entities and equivalent entities.

5 Algorithms and complexity

We now explore the practical computability of finding CF equivalences with their partial orders. To do this, we can take advantage of the efficient implementations of graph algorithms available in several tools. We use combinations of the following algorithms:

1. Constructing the internal representation of the digraph for the relationships *CR* and *CW*, such as the one of Fig.1. This is an immediate translation of the capability lists. The time complexity is $\boldsymbol{O}(|S|\times|O|)$, i.e. the product of the number of subjects by the number of objects.
2. Calculating the single object flow, see Fig. 2. This can be done by using depth-first search. The complexity of this algorithm is determined by the sum of the number of nodes plus the number of edges. For our application, this translates in time complexity $\boldsymbol{O}(|S|+|O|+(|S|\times|O|))$.
3. Finding the strongly connected components of the digraph obtained in Step 1, i.e. the CF-equivalent sets, together with their partial order, according to the principles of Sect. 3. Several efficient algorithms are known to do this. The one that we chose is Tarjan's algorithm [23].The complexity of this algorithm is linear on the number of edges plus the number of nodes, which in our case translates into $\boldsymbol{O}(|S|\times|O|+|S|+|O|)$.
4. To answer the global flow problem Step 2 is repeated for each object, so the complexity of this step is $\boldsymbol{O}(|O|\times(|S|\times|O|))$, or $\boldsymbol{O}(|O|^2\times|S|))$.
5. Transitive reduction. This is not indispensable but it leads to digraphs that are easier to read. The complexity of this algorithm in the case of acyclic digraphs is, once again, $\boldsymbol{O}(|S|+|O|\times|S|\times|O|)$.

In complexity evaluation, it is customary to consider only the dominant component, and so for each of the algorithms 1, 2, 3 and 5 the time complexity can be taken to be $\boldsymbol{O}(|S|\times|O|)$, which is polynomial, in fact quadratic if $|S|=|O|$. For the single object flow problem we use algorithms 1 and 2, leading to a total complexity of $\boldsymbol{O}(2\times(|S|\times|O|))$. For the global object flow problem, we use algorithms 1, 3, 4, 5, leading to a total complexity of $\boldsymbol{O}((3\times(|S|\times|O|))+(|O|^2\times|S|))$. Applied to the example of Section 4, this sequence of algorithms leads to the digraph of Fig. 3.These complexity orders could be refined, for example we have not considered the fact that Step 3 can reduce significantly the number of subjects and objects, compare Fig. 1 with Fig. 3.

For the purpose of this paper, the important conclusion is that only polynomial-time algorithms are needed for our method: in complexity theory, such algorithms are considered to be *efficient*.

6 Simulation results

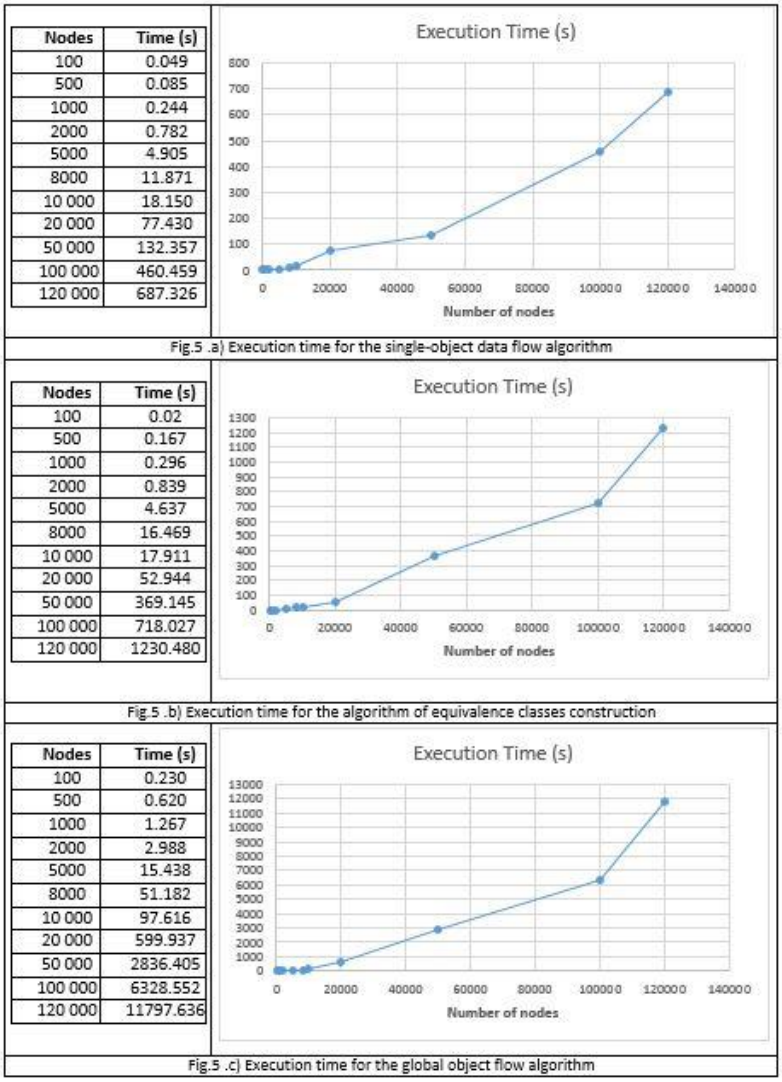
The complexity analysis that we have done so far describes upper bounds on the growth of functions depending on the number of entities but tells little about real execution times. Further, upper bounds are very seldom reached and good implementations apply many optimizations. It is then useful to perform simulations to have realistic estimates. We ran a number of simulations in MATLAB [9] to check real execution times. We used an Intel Xeon Processor E5-2603 v4 with 6 cores, 1.7GHz and 64 GB of RAM, in a DELL Precision Tower 7910 customized with hardware and software for heterogeneous processing. This is a fairly powerful system for research applications.

For each size considered, a network was generated with a random Boolean capability list for each subject. The ratio of numbers of protected objects per subject can vary greatly among types of systems, but normally

there are many more objects than subjects. Supposing application to RBAC, and following the advice of an expert [2] we considered networks where there are 24 objects for every subject, which means that, for an organization of size 100, we have 4 subjects and 96 objects. However, the simulation times do not change much if we change these ratios, e.g. they are similar if we assume that there are equal numbers of subjects and objects.

The execution times are shown in Fig. 5. Fig. 5.a) shows the times for Steps 1 and 2 in Section 5, Fig. 5.b) shows the times for Steps 1 and 3, and Fig. 5.c) shows the times for Steps 1,3,4,5, leading to representations such as the one of Fig.3. We see that the curves confirm our complexity analysis of polynomial times. The number of nodes is the number of entities.

For Fig. 5.c), our program hit a memory limit at approximately 120,000 nodes after about 196 minutes and for comparison we terminated the simulation at the same number for the other two simulations. Since the algorithms are polynomial, calculation times will improve with more powerful computers.



Nodes	Time (s)
100	0.049
500	0.085
1000	0.244
2000	0.782
5000	4.905
8000	11.871
10 000	18.150
20 000	77.430
50 000	132.357
100 000	460.459
120 000	687.326

Fig.5 .a) Execution time for the single-object data flow algorithm

Nodes	Time (s)
100	0.02
500	0.167
1000	0.296
2000	0.839
5000	4.637
8000	16.469
10 000	17.911
20 000	52.944
50 000	369.145
100 000	718.027
120 000	1230.480

Fig.5 .b) Execution time for the algorithm of equivalence classes construction

Nodes	Time (s)
100	0.230
500	0.620
1000	1.267
2000	2.988
5000	15.438
8000	51.182
10 000	97.616
20 000	599.937
50 000	2836.405
100 000	6328.552
120 000	11797.636

Fig.5 .c) Execution time for the global object flow algorithm

Fig. 5. Simulation times with MATLAB

Several types of context-dependent optimizations can be envisaged. Different objects can be treated as one if they have the same access control lists, which can be the case in many organizations where we have object categories such as financial, personnel, etc., with the same access control list for all objects in each category, see the concept of *view* in OrBAC [14]. Or also, the techniques we propose could be performed

level-by-level within hierarchies of object categories. If this is done, the set of objects could be the (probably much smaller) set of object categories instead of the set of all objects in the organization. Likewise, the number of subjects can be significantly reduced by considering roles instead of subjects. Yet other optimizations may be possible if, as it often happens in practice, a large network can be partitioned in several partially independent networks. Fig. 5 shows how much better execution times are if the number of entities is reduced by a factor of 10. This is a subject of further study, namely for applications to social networks.

7 Relation between CF and CHS-equivalence

It would be interesting to be able to conclude that the two equivalences we have worked with are identical, this would mean that by looking for CF-equivalences one can find *all* CHS-equivalences and vice-versa. Unfortunately this is not true, but only because of very special cases. To see this, consider a very simple network with one object *O* and two subjects *S1* and *S2*, and:

CR(S1,O), CR(S2,O)

We have: *CHS(O)=CHS(S1)=CHS(S2)={O}:* the three entities are not CF-equivalent and so our method cannot find that they are CHS-equivalent. More generally, a useful result would be: *CF(e,e') iff CHS(e) ⊆ CHS(e')* but the example shows that this is false in the reverse direction.

Note that this difficulty is limited to one level only, since if *CW(S1,O1)*, then *CHS(O1)={O,O1}* which breaks the *CHS* equivalence.

This difficulty can disappear if the difference between subjects and objects is eliminated, if some conditions are imposed, or if the model is changed. We excluded the first possibility because we are addressing access control systems. A full discussion on this topic would lead to complexities that are extraneous to this paper, whose focus is on algorithms.

For the purpose of this paper, the conclusion to be retained from this section is that our method may fail to detect very few, if any, CHS equivalence sets and this may only lead to partial orders still valid, but slightly more complex than they need to be.

8 A larger example

The example of Section 4 was very small, to be easily understood. In this section, we give a larger example, but still small enough that it can be checked manually. The capability lists are given in Table 4 and the partial order, the equivalence classes and the areas in Fig. 6. As mentioned in Section 4, Fig. 6 defines a class of possible implementations of the flows defined in Table 4. As earlier, we have oriented the graph to show increasing secrecy levels.

Table 4. Capability lists for a larger example

S1	CR	O2,O8	S5	CR	O4
	CW	O2,O4,O6		CW	O9
S2	CR	O5,O10	S6	CR	O1,O3
	CW	O7		CW	O5
S3	CR	O5,O6,O8	S7	CR	O9
	CW	O7,O8		CW	O4,O9
S4	CR	None	S8	CR	O5
	CW	O3		CW	O3

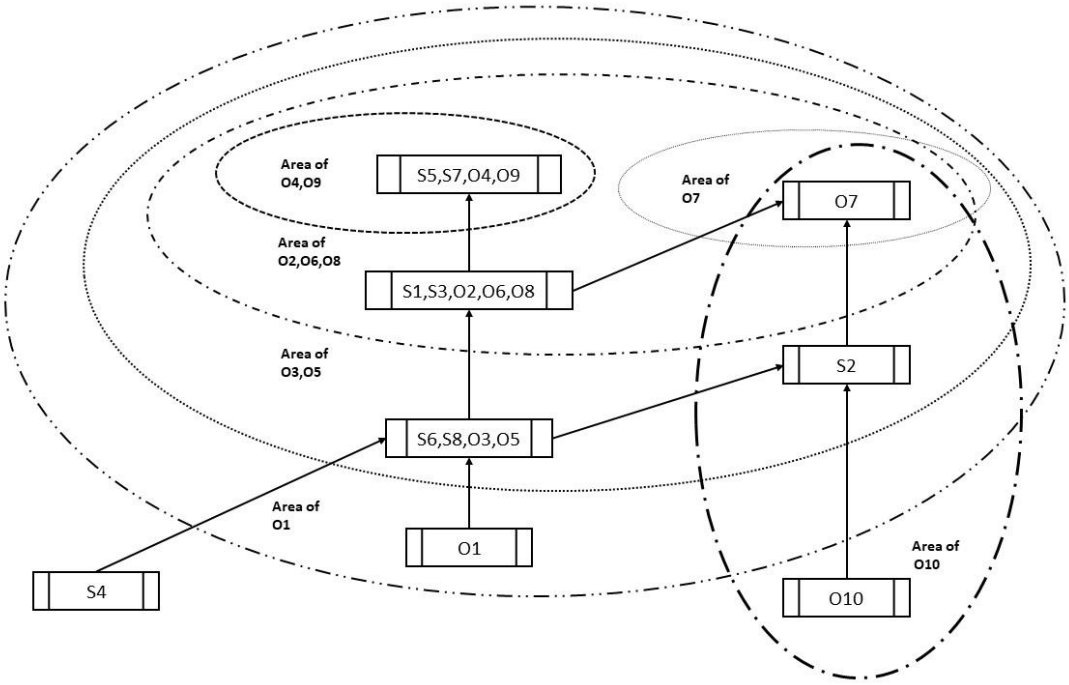


Figure 6. Equivalence classes, partial order and areas for the example of Table 4

9 Extension to RBAC and role engineering

A full discussion of RBAC is beyond the aims of this paper. We give here a summary of how some of the methods we have presented could be applied to some aspects of RBAC.

RBAC has capabilities that go well beyond access control to data, but has no specific mechanisms for data flow control. The unwanted data transfers can be controlled in RBAC indirectly by carefully designing roles, permissions and by static or dynamic constraints. However to do so, it is very useful to be able to answer our initial question, which is, where can data end up in a given RBAC configuration, in order to decide what changes need to be applied.

We limit ourselves to *core RBAC* without role hierarchies or constraints, and we consider only a single session, hence users and subjects can be identified. Only read and write permissions on data objects will be considered. So our previous *subjects* can be seen as *roles*. Permission lists for roles can be derived with these assumptions, thus the analyses that we have discussed so far can be applied to RBAC networks. See Saunders et al. [22] for a discussion on the relationship between RBAC models and the Access control matrix model. As an exercise, the reader may wish to apply our method to the RBAC example of [19]. If we assume 10 users per role, then there are considerable efficiency improvements as we can see from Fig. 5. If, in addition,

we assume that policies group objects with identical permissions, then there may be other significant improvements.

This leads to two applications of our method in RBAC, which can both be derived from observations presented in Section 4. The first is the following. If secrecy constraints exist, such as one by which the contents of certain objects can only be known to certain roles, then the proposed algorithms can be used to determine whether these constraints are violated. They can also be used to determine in which objects certain data should be put in order to be secrets of certain roles.

A second application is to role engineering. Role engineering for RBAC is the process of defining roles, user-role assignments, and permission-role assignments for a given organization [7]. The following facts can be important in role engineering:

- That certain roles can know no data, and so perhaps they can be eliminated: such would be a role corresponding to *S1*in Section 4 or *S4* in Fig. 6.
- That certain roles can know the same data as others, and so perhaps they can be merged, such is the case for roles corresponding to *S2, S4, S5* in the example of Fig. 3.
- That certain objects can store the same data, and so the objects and the permissions related to them can perhaps be merged, such is the case for objects *O2, O6, O8* in the example of Fig. 6.

The proposed algorithms provide such information. But the results of our method can only suggest changes to the RBAC model: the decision of implementing these changes must rest with the role engineer or the administrator, since reasons can exist not to do so.

As far as we know, we are the first to propose such methods for role engineering.

Several issues remain for further study. If sessions are introduced, then we must make assumptions on whether users can retain data from one session to another. Further, the feasibility of our technique in the presence of constraints should be studied. Papers [1][10] present other ideas that are well worth exploring from the point of view of available algorithms and simulation.

10 From capability lists to Label-based access control and Multi-level models

In its simplest form, a Label-based access control (LaBAC) model [4] can be defined by associating labels to subjects and objects, and using access control rules based on labels. We show how the concepts developed so far lead to LaBAC models, which define data flow controls that are identical to the ones defined by the capability lists from which they were generated. We write *s:lab(s)* or *o:lab(o)* to mean that subject *s* (object *o*) has label *lab(s)* (*lab(o)*). Given networks such as the ones of Fig. 3 or Fig. 6, the following method is sufficient:

1) Assign to each subject and object a label designating the set of the areas in which it finds itself.
2) The access control rules are:
    - *CR(s:lab(s), o:lab(o)) iff lab(o)* $\subseteq$ *lab(s)*
    - *CW(s:lab(s), o:lab(o)) iff lab(s)* $\subseteq$ *lab(o)*

For example, in Fig. 6, we have: *O10*:*{O10}* and *S2:{O1,O3,O5,O10}*, hence *CR(S2,O10)*. Clearly, the LaBAC model for this example defines a flow control that is identical to the flow control defined by the capability list of Table 4. The construction is generic and so for any capability list, an equivalent LaBAC model can be generated efficiently. Conversely, given the labels, corresponding roles can be defined for each subject and read and write permission lists can be generated for each role thus giving an elementary RBAC system.

One could create a correspondence between the labels above and more conventional security labels, e.g. *{O1,O2,O3,O4,O5,O6,O8,O9},*which is the label for all the elements in the top left equivalence class of Fig. 6, could be called *"TopSecret1"*, while *{O1,O2,O3,O5,O6,O7,O8,O10},* which is the label for *O7*, could be called *"TopSecret2"* and so on as desired.

LaBAC access control systems created in this way are essentially Mandatory access control systems or Multi-level systems, such as Bell-La Padula but based on partial orders rather than lattices.

11 Conclusions and future work

We have identified some data flow analysis problems in access control networks, and we have shown that they are practically solvable with well-known and efficient graph algorithms, using as input the capability lists of the subjects, which will be permissions lists in the case of RBAC. This finding has been validated by both algorithmic analysis and simulation. Simulation has shown that our method is usable for networks of several tens of thousands of subjects and objects; in fact, if we consider ongoing progress in the area of graph processing [18], possibly for any practically conceivable organization sizes. The method can be used to answer several important questions such as: if some secret data are stored in some database, who can be able to read them, directly or indirectly? For each subject or object, exactly what data can it have available? If certain data should be secret of certain subjects or objects, where should the data be stored? What subjects and objects are equivalent for the data they can be able to hold? What reorganizations are possible, without changing the data flows? Our concept of *area* of an object in a network seems to be new and useful, and so is the possibility of identifying secrecy levels in arbitrary (possibly RBAC) networks.

This paper can also be seen as an experimental confirmation of the principles presented in [17]. We have shown that the partial order of components in a data flow network can be seen as a hierarchy of secrecy levels. This generalizes the well-known concept of lattice flow model presented in [6]. It generalizes it for two reasons: partial orders are a less restrictive structure than lattices, and partial orders can be always found efficiently, by using our method, for any network of entities that is described in terms of reading and writing capabilities. Based on these concepts, one can imagine graphic interfaces that would make it possible to design systems with secrecy requirements by manipulating on the screen graphic representations such as the one proposed here. For scalability however, abstraction mechanisms such as encapsulation will have to be devised.

With respect to previous work, the contribution that is closest to ours is [1]. These authors describe powerful methods and tools that go much beyond what we have done, but don't provide the detailed algorithm analysis and simulations that we provide here, hence they provide no information on the sizes up to which their tools could be practical. In addition, although they introduce the concept of 'reduction by information equivalence classes', they do not introduce the generic secrecy level concepts that we have mentioned, based on the order-theoretical concepts of Section 3. Similar observations apply to [10] [11]. The authors of these three papers consider the general case where the permissions can change, but seem to miss the interesting conclusions that can be drawn when the permissions are fixed, as we do in this paper. The conceptual bases they use are also fairly different from ours. Reading of these papers raises many interesting questions at the intersection of these partially complementary approaches. These papers also give an idea in what kinds of tools our algorithms can be used.

Other topics for future work present themselves, for example, in connection with RBAC: extension to RBAC with static constraints, and using our method for RBAC auditing. Extension to Boolean conditions, dynamic constraints and changes in capabilities (such as by administrative action) are further beyond. The applicability of these methods in Internet of Things and Cloud networks is also on our research agenda.

Acknowledgment. This research was funded in part by the Natural Sciences and Engineering Research Council of Canada. We are grateful to Jurek Czyzowicz for discussion on graph algorithms, to Ahmed Lakhssassi for having allowed us to use his powerful computer, to Sofiene Boulares for discussion on the theory and practice of access control methods, and to Mustapha Benmahbous of Xpertics for having shared with us some views based on his long experience in role engineering. The anonymous referees have contributed to the clarity and completeness of this paper.

References

1. P. Amthor, W.F. Künhauser, A. Pölk. WorSE: A workbench for model-based security engineering. Computers & Security 42 (2014), 40-55.
2. M. Benmahbous. Personal communication, 2017.
3. J. Barkley. Comparing simple role based access control models and access control lists. Proc. of the 2nd ACM Worksop on RBAC, 1997, 127-132.

4. P. Biswas, R. Sandhu, R. Krishnan. Label-Based access control: An ABAC model with enumerated authorization policy. Proc. 2016 ACM Intern. Workshop on Attribute Based Access Control (ABAC 2016), 1-12.
5. R. Chon, T. Enokido,V. Wietrzsk, and M. Takizawa. Role locks to prevent illegal information flow among objects. Proc. of IEEE 18th Intern. Conf. on Advanced Information Networking and Applications (AINA-2004), Vol. 1, 196-201.
6. D.E. Denning. A lattice model of secure information flow. Comm. ACM 19(5), 1976, 236-243.
7. D.F. Ferraiolo, D.R. Kuhn, R. Chandramouli. Role-based access control. 2nd Ed. Artech House, 2007.
8. R. Fraïssé. Theory of relations. North-Holland, 1986.
9. A. Gilat. *MATLAB: An Introduction with applications.* 2nd Ed. John Wiley & Sons, 2004.
10. M. Gofman, R. Luo, J. He, Y. Zhang, P. Yang, S. Stoller. Incremental information flow analysis of role based access control. In: International Conference on Security and Management, 2009, 397–403.
11. M. Gofman, R. Luo, J. He, Y. Zhang, P. Yang, S. Stoller. Rbac-pat: A policy analysis tool for role based access control. Intern. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2009), LNCS 5505, 46-49.
12. F. Harary, R.Z. Norman, D. Cartwright. *Structural models: an introduction to the theory of directed graphs.* Wiley, 1965.
13. K. Izaki, K. Tanaka, M. Takizawa. Information flow control in role-based model for distributed objects. Proc. 8th Intern. Conf. on Parallel and Distributed Systems (ICPADS 2001), 363-370.
14. A.A.E. Kalam, R. E.I. Baida, P.Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, A. Miege, C. Saurel, G. Trouessin. Organization based access control. In Policies for Distributed Systems and Networks, Proceedings. POLICY 2003. IEEE 4th International Workshop on, 120-131.
15. B.W. Lampson. Protection. In: Proc. 5th Princeton Conf. on Information Sciences and Systems (1971), 437-443.
16. L Logrippo, Logical method for reasoning about access control and data flow control models. Proc. of the 7th Intern. Symp. on Foundations and Practice of Security (FPS 2014), LNCS 8930 (2015), 205–220.
17. L. Logrippo. Multi-level access control, directed graphs and partial orders in flow control for data secrecy and privacy. Proc. of the 10th Intern. Symp. on Foundations and Practice of Security (FPS 2017), LNCS 10723 (2018), 111-123.
18. MIT News. Device allows a personal computer to process huge graphs. http://news.mit.edu/2018/device-allows-personal-computer-process-huge-graphs-0531. Accessed May 4, 2018.
19. S. Nakamura, D. Duolikun, A. Aikebaier, T. Enokido, M. Takizawa. Synchronization Protocols to Prevent Illegal Information Flow in Role-based Access Control Systems. Proc. of International Conference on Complex Intelligent and Software Intensive Systems (CISIS-2014), 279-286.
20. S.L. Osborn. Information flow analysis of an RBAC system, Proc. of the seventh ACM symposium on Access control models and technologies, (SACMAT 2002), 163-168.
21. P. Samarati, E. Bertino, A. Ciampichetti, S. Jajodia. Information flow control in object-oriented systems. IEEE Trans. On Knowledge and Data Eng., 9(14), 1997, 524-538.
22. G. Saunders, M.Hitchens, V.Varadharajan. Role-based access control and the access control matrix. In: 5th Intern. Conf. on Inform. and Communic. Engg. (ICICS 2003), LNCS 2836, 145-157.
23. R. E. Tarjan. Depth-first search and linear graph algorithms, SIAM Journal on Computing, 1(2) (1972), 146–160.
24. C. N. Zhang, C. Yang. Information flow analysis on Role-based access control model. Information Management & Computer Security, 10 (5), 2002, 225-236.