Function-Based Access Control (FBAC): From Access Control Matrix to Access Control Tensor*

Yvo Desmedt
The University of Texas at Dallas, TX, USA
University College London, London, UK
Yvo.Desmedt@utdallas.edu†

Arash Shaghaghi
UNSW Australia, Sydney, Australia
Data61, CSIRO, Australia
A.Shaghaghi@unsw.edu.au†

ABSTRACT

The misuse of *legitimate* access to data is a serious information security concern for both organizations and individuals. From a security engineering viewpoint, this might be due to the failure of access control. Inspired by Functional Encryption, we introduce Function-Based Access Control (FBAC). From an abstract viewpoint, we suggest storing access authorizations as a three-dimensional tensor, or an Access Control Tensor (ACT) rather than the two-dimensional Access Control Matrix (ACM). In FBAC, applications do not give blind folded execution right and can only invoke commands that have been authorized for function defined data segments. So, one might be authorized to use a certain command on one object, while being forbidden to use the same command on another object. Such behavior can not be efficiently modeled using the classical access control matrix or achieved efficiently using cryptographic mechanisms. Here, we lay the theoretical foundations of FBAC and summarize our extended work on implementation and deployment recommendations.

Keywords

Access Control; Access Control Matrix (ACM); Access Control Tensor (ACT); Function-Based Access Control (FBAC)

1. INTRODUCTION

Access control models derived from Access Control Matrix (ACM) encompass three sets of entities, Subjects, Objects and Operations. Typically, objects are considered to be files and operations are regarded as Read, Write, and Execute. This implies an 'open sesame' approach when granting access to data, i.e. once access is granted, there is no restriction on command executions. The aforementioned condition is directly associated with one of the prevalent threats affecting organizations, the Insider Threat. To understand the limitations of existing access control systems, we start with a brief revision of two prevailing incidents, the Wikileaks and HMRC cases.

After the September 11 terrorist attacks on US soil, government agencies in the United States began allowing a greater sharing of information as a defense procedure against future terrorist strikes. This included sharing of confidential information between the US Department of State and the US Department of Defense. However, in 2010 after a massive leak of diplomatic cables by Manning, a low-ranked personnel of the Army, the US Department of State revoked this access, to prevent further leaks. Manning did not break any system and used credentials to access highly classified information. Unbelievably, to share this information with WikiLeaks, all that had to be done was to copy this information to a CD drive and take it home – causing the largest leak of military and diplomatic cables in US history. Similarly, the Her Majesty's Revenue & Customs (HMRC) incident involved the loss of five million UK national records by an employee who copied the confidential data onto disks and sent it through the post.

As implied, the main reason behind these incidents is that once the user is granted authorization to access data, s/he has the full authority on how to to use it. Evidently, as with the case of access revocation to the US Department of Defense, removing access is not a remedy for this type of security threat. Neither is requiring high-security clearance for every officer/employee or enforcing strict limitation, as all of these prevent an organization performing its usual tasks. We argue that we need models and mechanisms that allow *an authorized entity to perform required operations on confidential information but not have full access to it.* As a simple example, a Department of Homeland Security (DHS) agent should be able to search in confidential information but s/he should only see the relevant information and not be able to run any other operation on it, such as copy and print. At the same time, to ensure information flow control, access restrictions should be applied at the lowest possible level, i.e. data block. Indeed, our ideas are not restricted to text and also apply to images and videos. For example, even when viewing an image - we consider this as being a write operation on the device "screen" - only the relevant parts of an image must be shown with the non-authorized parts blurred.

We have thoroughly reviewed existing access control models in [2]. In brief, Access Control Matrix (ACM) the core concept behind currently implementable systems predates the Internet, computer viruses and massive hacking. In fact,

*The first full version of this paper is available at http://arxiv.org/abs/1609.04514.

†Both authors contributed equally to this work.

MIST'16, October 28 2016, Vienna, Austria

DOI: http://dx.doi.org/10.1145/2995959.2995974

at that time of conception, computers had limited resources, and there were very limited number of applications. Today, however, there are a huge number of applications on each platform with a massive number of functionalities. Therefore, information flow control mechanisms that rely on entropy to quantify information flow are not reliable by themselves as entropy does not measure the value and the importance of data.

Here, inspired by Functional Encryption [1], we introduce Function-Based Access Control. From a foundation viewpoint we replace the access control matrix with an Access Control Tensor (ACT), which in effect is a generalization of an access control matrix. In FBAC, objects are data blocks and functions are the commands available in applications, such as Copy/Paste and Search. In the policy specifications of FBAC, the commands may be defined as standard — as we know them today, or restricted. For example, the Copy/Paste commands could be custom defined such that when a researcher quotes a part of the text that has a citation, both the text and citation are copied together to the destination. Or, the email function could be customized such that when a sensitive part of a document is emailed, the supervisor is always copied. Moreover, in FBAC, protected objects *do not have to be predefined*, and the function can be customized to protect objects that are created on the fly. In our proposed access control model, *applications do not have blindfolded execution right and can only invoke commands that have been authorized for data segments in respect to subjects*. To the best of our knowledge, FBAC is the only access control model capable of supporting this level of precision. In Section 2, we present an overview of FBAC's theoretical foundations. We then include a succinct report of our prototype implementation and conclude with our current work-in-progress.

2. FUNCTION-BASED ACCESS CONTROL (FBAC)

With **Functional Encryption** [1], given an encrypted text of a certain plaintext, one can compute from the ciphertext f(Plaintext), where f is an authorized function, without revealing anything additionally about the rest of the plaintext. As an example, using this tool one could "search" whether a certain string is (or not) on encrypted data without decrypting it. This is in sharp contrast with how access to data is being controlled today. Indeed, a person searching for the word "terrorist" in a file, must have received read permission for the file and execute permission for the program that does the search, which gives him/her full access to the whole file!

The main challenges for using functional encryption is to construct a system that supports the creation of keys for any function in both public and non-public index settings and is efficient. Overall, although promising, functional encryption is still in its infancy and much further practical and theoretical advancement is required to solve associated open problems. Therefore, we propose FBAC as an alternative access control solution to achieve a similar level of control over data.

2.1 A First Definition

The current approach in access control finds its foundations in the 1974 paper by Lampson [4] and formalized in 1976 by Harrison-Ruzzo-Ullman [3]. Its main limitation, from our perspective, is that it has only two dimensions, being, one dimension corresponding with objects and one with subjects. In our definition, we will use a three-dimensional approach and use "function" as the third dimension.

In our definition, an object could correspond, with a file, an XML record, as data in a register, etc. Moreover, functions could be at the level of the operating system (such as grep), but also an operation inside an application (such as search used inside a browser, an editor or an e-mail reader).

Before giving our actual definition, we note that the number of inputs to a function depends on the function. Our definition has to take this into account. Moreover, not all inputs to a function are "predefined," as we now explain. Consider grep. Usually *grep* operates on a *file* and a *pattern* is given, e.g., from the terminal. Moreover, grep has several options, such as "quiet," which makes grep output a Boolean. We do *not* regard the "pattern" and the options as objects. We will explain later how to deal with these non-object inputs.

To deal with the fact that a function can have more than one object as input (such as Copy/Paste), we introduce the following definition.

DEFINITION 1. *When O denotes the set of object, we let $O^1 = O$ and recursively we define $O^j = O^{j-1} \times O$ $(j \geq 2)$. Moreover, we let $O^0 = \emptyset$. We also define*

$$O^* = \bigcup_{j \geq 0} O^j.$$

We now define a first version of Access Control Tensor (ACT).

DEFINITION 2. *Let S be the set of subjects, F a set of functions, O^* as defined earlier. The three-dimensional table A is a mapping from $S \times F \times O^* \rightarrow \{False, True, N/A\}$. When $f \in F$ has n objects as input, $o \in O^*$ is an m-tuple, s is a subject, then $A(s, f, o) = N/A$ when $m \neq n$. If $m = n$, and $A(s, f, o) = True$ then subject s can execute the function (command) f on object o, else the subject can not. We call A the access tensor. We call (S, F, O, A) an elementary function-based access control, or E-FBAC.*

Evidently, the set {N/A, False, True} could be replaced by {N/A, Forbidden, Authorized}.

One could observe that the typical entries to the Access Control Matrix (ACM), such as read and write, do not appear in our ACT. The reason for this is that our functions that can read cannot write. Moreover, every read only function can be regarded as writing to standard output. So, the function, or the input parameters of the function, will define that aspect. Note that each command inside an app, such as an editor, is regarded as a function and falls under above access control.

2.2 The Main Definition

The elementary function-based access control is too primitive for many different reasons. Let us reconsider *grep* and assume we allow a user in Homeland Security to search files in the CIA for the word terrorist. Using the grep option "context=NUM" and using a very large value for NUM, the user will be able to access the complete file, which might not be the purpose. Moreover, the user could use grep to search for other keywords (or in general patterns) than the word terrorist. We first discuss how we could fit such restrictions in E-FBAC.

Suppose we define a new command grep_terrorist_count=5, which only allows the user to search in files for the word terrorist and which prints 5 lines of context. In other words, this command has no other options. Then controlling access when using grep_terrorist_count=5 can be described using the E-FBAC approach. Obviously, in practice we want the user to have the flexibility to use options, which we now address.

DEFINITION 3. *Let S be the set of subjects, F a set of functions, O^* as defined in Definition 1. The entries to the three-dimensional table A with dimensions identified by S, F, and O^* are of the type "False", "True$[P_{(s,f,o)}]$," and N/A. When $f \in F$ has n objects as input, $o \in O^*$ an m-tuple, s a subject, then $A(s,f,o) = N/A$ when $m \neq n$. When $m = n$, and $A(s,f,o) =$ False, the subject can not execute the function (command) f on object o. In the other case $[P_{(s,f,o)}]$ is an option. If the option is specified, then the predefined program $P_{(s,f,o)}$ comprises the joint list of options (with their parameter) together with the standard input. If P returns True, then the function f with the aforementioned list of options and standard input can be executed by s on o. We call A the access tensor. We call (S,F,O,A) a generalized function-based access control, or G-FBAC.*

Note that using G-FBAC in practice might make access control very slow. We suggest instead to replace $P_{(s,f,o)}$ by a regular expression. If the list of options and the standard input satisfies the regular expression, f with the restrictions indicated in Definition 3, can be executed. We call this approach a *regular-expression function-based access control*, or in short RE-FBAC. Evidently, our approach is very different from the one giving subject execution right to functions (or operations) and read/write to objects. Indeed, whether an operation can be executed or not is object dependent. To emphasize this aspect of our approach, we call this *the Function-Data Granularity, or the F-D granularity.* It allows to specify that a user can only use "grep" with very restricted options and patterns on outside data, but allowing grep in an unrestricted way on his/her own data.

Before we proceed further in this section, let us make some preliminary observations. As is well known, any three dimensional table can be mapped into several two dimensional tables. Indeed, for each (subject,object) we could specify which functions could be executed, and provide above restrictions specified by $P_{(s,f,o)}$. However, anyone familiar with Lampson's approach immediately observes that this does not match the Lampson's description and one also looses the deeper insight the third dimension brings.

Arguably, the classical Attribute-Based Access Control for XML does not support a matching level of control over data. Indeed, XML organizes the document into "records." When granting read access to this record, the *maximal output* a user can see is the whole record. When applying FBAC to an XML document or any other type of file, the *maximal output* a user can see, can contain significantly less data than the full record. Finally, the power of FBAC in non-textual contexts will be illustrated in the proof of implementation (See Section 3).

2.2.1 Further Output Controls

We first note that in certain contexts it still makes sense to define customized versions of classical commands, such as grep. Indeed, a customized command could further restrict the output by blanking out words or sentences containing predefined words such as "submarine." Unix allows the use of "pipe" (i.e., |), which from a mathematical viewpoint correspond to a composition of functions, e.g., f after g. To regulate access to the use of pipe, we could regard $f \circ g$ as a new function and then control this as above. We now discuss an alternative approach.

If we want to allow the use of pipe and want to avoid having to deal with specifying all possible combinations of compositions – note that the number of different functions one can define with a given finite domain is finite, but too large to have practical value. The following approach, which we illustrate with grep, could be used.

Grep can be executed on files, but also on standard input, the latter enabling to use grep on an output of a prior command when using pipe. In our approach the latter use of grep corresponds with a case in which grep has no predefined object as input. That implies that we can regard grep as being two commands, one being grep_in_file and grep_in_standard.

The first has one predefined object as input; the second has zero. In the latter, the restriction on the standard input will then be specified by the option P, as defined in Definition 3.

2.3 Access Control Tensor (ACT) in Practice

Storing rights in an access control matrix is often too impractical or would slow down enforcement. Several approaches have been used. Some of these are policy dependent, such as the Unix concept of having the *user* (owner), *group(s)*, and *world*, when dealing with access control to files. From a conceptional viewpoint, this policy corresponds with a compressed authorization list per object. We now wonder what the equivalent ones are when using an access control tensor.

In the classical approach, an authorization list corresponds to a column in the access control matrix. In other words, given an object, we obtain this list. Since our approach is 3-dimensional, given solely an object, the rights described related to that object are 2-dimensional, and so it can no longer be called a list. We therefore call this an *authorization matrix*, i.e., for a given object(s) o the authorization matrix gives $A(s_i, f_j, o)$, i.e., all values $A(s_i, f_j, o)$ for all i and all j. Obviously, we can compress this matrix by only considering functions for which $A(s_i, f_j, o)$ will be different from N/A.

In operating systems, capabilities play an important role. In our setting this will be 2-dimensional and we talk about *capability matrix*, or just *capability*. For each fixed subject s we can have a capability corresponding to the matrix $A(s, f_i, o_j)$, which contains these values for all i and all j. Obviously, we can compress this matrix by only considering pairs of (functions,objects) for which $A(s, f_i, o_j)$ will be different from N/A.

Obviously, we will have a new two-dimensional control mechanism, which when given a particular function will reveal which subject have rights to which objects. Since this matrix has the same dimensions than in the classical case, we call this matrix an *access control matrix*. In other words, for each fixed function f we can have an access control matrix corresponding to the matrix $A(s_i, f, o_j)$, which contains these values for all i and all j. Obviously, we can compress this matrix by only considering object(s) for which $A(s_i, f, o_j)$ will be different from N/A.

When systems are large, storing above matrices may be impractical. Moreover, when we are using a particular application, only the commands (functions) that are available for this application are relevant. In such circumstances, we will have two inputs, such as $(\text{subject}, \text{object}) = (s, 0)$, and want to know the rights to all (or a subset) of functions. We call $A(s, f_i, o)$ given the values for all i, a *function list*. Obviously, we can perform the aforementioned N/A compression. If we have an application P and we want to restrict the function list to the application, we write $A_{|_P}(s, f_i, o)$ to indicate that f_i is a function available in the application P and speak of *application restricted function list*. Note that we can regard the commands available in a terminal application, as $P = OS$ or $P = \text{terminal}$.

For security audits it might be useful to know who has access to a certain object o when using a function or command f. We call such a list a *subject list* and when given (f, o) it gives $A(s_i, f, o)$ for all i. When we have a distributed system, we could restrict the subjects to $T \subseteq S$. We denote this restriction as $A_{|_T}(s_i, f, o)$. (We silently assume that (f, o) is a meaningful pair.)

Finally, when given (s, f) we want to know on what objects the subject s can execute f and with what restrictions. We call the corresponding list an *object list* it gives $A(s, f, o_i)$ for all i. When we want to restrict the list of objects to $B^* \subseteq O^*$, we have $A_{|_{B^*}}(s, f, o_i)$. B^* may correspond to object(s) inside a certain directory, or objects owned by a certain organization, etc.

The above concepts can be used for all our variants of FBAC, i.e., E-FBAC, G-FBAC, RE-FBAC.

3. FBAC IN PRACTICE

Policy: In a lattice based information flow policy such as Bell-LaPadula and Biba, we have a set SC of security classes and a relation $\preceq$ on SC such that $(SC, \preceq)$ is a lattice. With FBAC, in the case of confidentiality, given objects x and y and their corresponding security classes $\underline{x}$ and $\underline{y}$, information can flow from x to y if $\underline{x} \preceq \underline{y}$. In Biba's model, when s is a subject and o is an object, s can write when $\underline{o} \preceq \underline{s}$, where the $\underline{s}$ is the integrity level of s and similarly for $\underline{o}$. In general, with FBAC, we have *function dependent security classes* and in practice, we will specify these for subsets of functions. For an extended discussion on advantages of our function dependent policy and non-lattice based access control policies refer to [2].

Data Storage: One of the main requirements of implementing FBAC is the proper storage of data and authorizations. We store data in a purpose built data structure format, called Atomic Document. Such a document is composed of one or more Atoms – the smallest segments of a document, which is *indivisible*. These could be paragraphs in an unstructured document, a sub-tree in a tree-structured data, etc. Atoms have an accompanying policy, which *once executed for a subject* returns a *Function List* (F) — the policy is an Authorization Matrix but when we regard the matrix for one specific subject then it becomes a *Function List*. An Atom, within an Atomic Document, may be *Single* or*Linked*. In the latter, execution of a function affects one or more other atoms. For a formal definition of an Atomic Document along with supported features such as a classification see [2].

Prototype Implementation: Copyright is a serious concern for the right holders who are seeking innovative mechanisms to prevent it (and not just detect it). We have developed *Smacs*, an editor that enforces Function-Based Access Control, which if used properly could be an effective prevention mechanism against plagiarism. For it to be practical, we assume *all documents* are stored in Atomic format and are accessed using the developed editor.

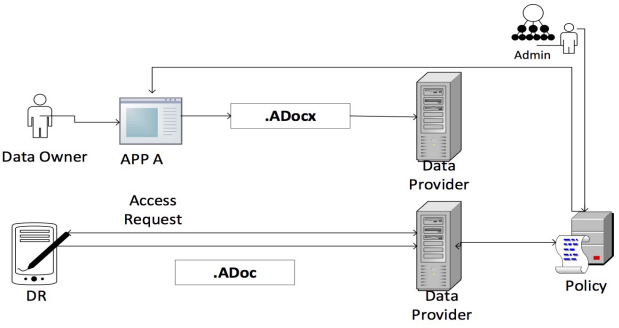

Figure 1: One possible Smacs deployment scenario.

Smacs, or Secure Emacs, is built on top of the GNU Emacs. In Smacs, the access control tensor is implemented as an Authorization Matrix (see Section 2). Documents are written in LaTeX format and converted into Atomic format with the authorization policy array attached as a separate file (*.ADocx*). The set of permissions an author may use for the atoms s/he creates is defined by the supervising administrator (see Figure 1). Whenever an access request is sent to the reference monitor, the array is processed and retrieves a regular expression characterizing the function (*.ADoc*). We have implemented all of the typical commands of a word processing software for the editor as well as support for both text and images. For example, in the case of images, when the Viewer is not authorized to view an image, the image may be hidden, blurred or shown with a watermark. Alternatively, when copying a cited text both the script and its citation are imported to the destination to prevent plagiarism. An extended discussion on the deployment of FBAC and lessons learned from developing Smacs is available in [2].

4. CURRENT WORK

We are currently extending FBAC at the level of operating system and further exploring compatibility requirements with existing access control models already used in organizations.

5. REFERENCES

[1] D. Boneh, A. Sahai, and B. Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography*, pages 253–273. Springer, 2011.

[2] Y. Desmedt and A. Shaghaghi. Function-Based Access Control (FBAC): From Access Control Matrix to Access Control Tensor. *arXiv preprint arXiv:1609.04514*, 2016.

[3] M. A. Harrison, W. L. Ruzzo, and J. D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[4] B. W. Lampson. Protection. *ACM Operating Systems Review*, 8(1):18–24, January 1974. Also in, Proc. 5th Princeton Symposium of Information Science and Systems, 1971.