Route-based authorization and discovery for personal data

Yousef Amar
Queen Mary University of London
y.amar@qmul.ac.uk

Hamed Haddadi
Queen Mary University of London
hamed.haddadi@qmul.ac.uk

Richard Mortier
Univeristy of Cambridge
richard.mortier@cl.cam.ac.uk

***Abstract*—When faced with systems in which third party components need to advertise the availability of data they gather, while other such components need to access it, solutions for delegated authorisation and discovery APIs for interoperability are needed. This work explores possible solutions, and converges on a testable implementation.**

I. Introduction

The rise of ubiquitous sensing via mobile and IoT devices has led to a surge in generation and collection of personal data. Meanwhile, concerns over privacy, trust, and security are becoming an increasingly important theme as different stakeholders attempt to take advantage of this data.

The Databox [2] – a set of networked services enabling individuals to manage their data and to provide other parties with controlled access to their data – seeks to remedy these issues. Databox systems have to be privacy-preserving by design, while allowing third-party processing and analytics to run on a user's personal data.

This poses unprecedented systems design challenges to which the solutions are of immense value both inside and outside the Databox paradigm. In such a system, a solution for delegated authorisation and APIs for interoperability are needed. This work aims to find these solutions and evaluate them against alternatives.

II. Research Context

A Databox is a home IoT hub, that is supported by cloud services. It makes diverse personal data sources (from online/-social to IoT to mobile) accessible and provides runtime APIs for interfacing with that data.

For both technical and privacy reasons, data processors are incentivized to access and process all data within the sandbox environment that is a Databox, and only emit results to the outside world. By bringing the analytics to the data, rather than the data to the analytics (in the cloud), Databox reduces the risk of inferences – let alone privacy breaches – to an exceptional degree.

Databox components run in isolated containers with limited, encrypted communication. Next to components that perform administerial functions, there are three core components.

Drivers, which can be developed by third parties, interface with outside data sources and query data. These write data into *stores* which provide a common API and access control for other components to access this data. Stores are system components and launched alongside drivers. Finally, *apps* (most of which will be written by third-party developers) can be launched with permissions to access certain stores, and thus certain data. Once these have finished their processing, they may be allowed to emit limited data to the outside world through an export service. Apps are packaged alongside a manifest file which, among other metadata, lists the required and optional permissions that an app may need.

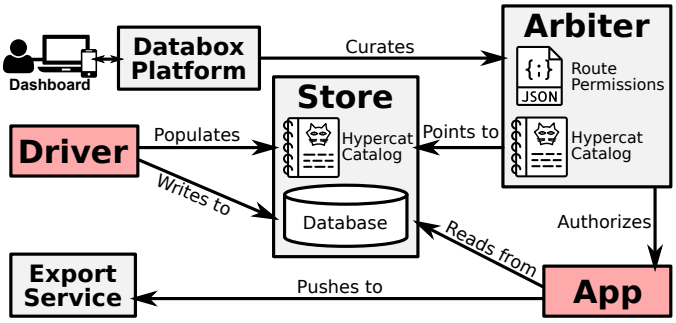

Fig. 1. Relationships between system and third party (red) components

III. Research Questions

It is necessary that within the Databox ecosystem, mechanisms for discovery, authorisation, and granular permission management be in place. Within the scope of the Databox ecosystem, we have two main problems, given that third-party drivers may make arbitrary data sources available that provide data in a variety of formats.

The first is how to dynamically catalogue ephemeral API endpoints (for both data and other interaction) and the metadata associated with these, such as type, format, additional granularity restrictions, and even custom vendor-specific metadata.

The second is how to manage and delegate the management of permissions and access control to these individual endpoints/items.

IV. Implementation

This section describes our current working implementation of our system. It combines a route-based permissions scheme, with a resource description standard (Hypercat [3]) and signed tokens (Macaroons [1]) for delegated authorization. We call the system component that performs these tasks the *arbiter*.

As a system that needs to be privacy preserving by design, all permission and access are most restricted by default. Only through user interaction should these permissions be able to be made laxer. For components that are written by third-party developers (highlighted red in figure 1), a number of precautions are already taken, such as isolating these on separate bridges, and having all source code undergo a rigid scrutineering process.

There are three core features of our system for resource description and authorization:

1) A root/top-level Hypercat catalogue is used to point to store-hosted catalogs to facilitate discovery and metadata retrieval by walking catalogs.
2) Distinct, granular macaroons are attached to any requests made to store endpoints that allow stores to independently determine whether a request is authorized or not.
3) Route-based permissions embedded in macaroons by the arbiter automatically control what operations an app or driver has been granted permission to execute, as long as the operation can be expressed as a route. This can be interfacing with data, or carrying out other functions explored below.

The relations between system components in this context are depicted in figure 1. The arbiter hosts a directory of all stores in the form of a Hypercat catalog. Our route-based permissions system applies to interfacing with Hypercat catalogs as well.

Macaroon caveats are conditions that must all be satisfied for the token to be considered valid. As such, using these would seem unsuitable for interleaving optional – or even mutually exclusive – permissions expressed as conditions, prima facie. This is however not the case, as individual caveats can be anything, including whitelists.

We therefore treat a set of caveats as a "product of sums", in the sense that each caveat is a conjunct that is ANDed with all other caveats, while individual caveats may define disjunctions in the form of whitelists with each contained disjunct ORed. Crucially, any combination of conditions can be expressed in this manner, making our extension of macaroons flexible enough to cover any expression of permissions.

The most critical caveat for our system of authorisation is one such conjunction: the *path* caveat of a route. We define a route simply as a combination of *target*, *path*, and *method*. Paths are strings, or string whitelists, with some optional formatting. They define accessible endpoints under a specified method for a single target. Methods are HTTP verbs that are generally mapped to CRUD operations, e.g. GET and POST to read from and write to a store respectively.

Figure 2 is an example of a set of route caveats, to illustrate the flexibility of route-based permissions in the context of Databox APIs. In this case, we describe the extent of a bearer's permissions in a considerably detailed manner. For illustration purposes, these are all encoded into one token; in practice, we would separate the paths out one per token. For instance, the bearer of a token encoded with the above caveats can:

- Access the (potentially censored or filtered) second-level store catalogue of the target store

```
target = databox-mobile-store
method = GET
path = [
  "/cat",
  "/ws",
  "/profile/kv",
  "/accelerometer/ts/*",
  "/gps/ts/latest",
  "/logs/*/ts",
  "/(sub|unsub)/light/ts/*"
]
time < 1490790593391
```

Fig. 2. An example of route caveats

- Connect to a target store's WebSocket notification server
- Access a specific store-hosted JSON document with the key "profile" in a key-value database
- Access all "accelerometer" time-series endpoints
- See only the latest readings from store data source "gps"
- Read any logs as long as they are for time-series data
- (Un)subscribe to notifications for any new "light" data

Any operation that can be made into a REST-ful endpoint is automatically covered by this permissions system, without any extra configuration, making it exceedingly future-proof. As already alluded to, by separating routes, it is also directly compatible with existing APIs and can map directly onto them. Permissions can be made more granular by adding additional general caveats (such as an expiry timestamp) or endpoint-specific caveats (such as a bounds for time-series data, or a destination whitelist for the export service).

V. Evaluation and Future Work

The question with the most practical importance to this system is whether or not it scales well, as compared to other techniques for discovery and authorization. We assert that our model of distributed discovery and access control is the most optimal, as the arbiter is only queried to mint new, short-lived tokens or to list the root store catalogue.

We plan to perform empirical tests to measure general performance metrics on typical hardware, identify any bottlenecks, and compare these with other techniques. As more and more apps, drivers, and stores are launched, another interesting question surfaces: can one extend token expiry lifespans to reduce load, and if so, at what cost in security, if any?

References

[1] Arnar Birgisson et al. "Macaroons: Cookies with Contextual Caveats for Decentralized Authorization in the Cloud." In: *NDSS*. 2014.

[2] Amir Chaudhry et al. "Personal data: thinking inside the box". In: *Proceedings of The Fifth Decennial Aarhus Conference on Critical Alternatives*. Aarhus University Press. 2015, pp. 29–32.

[3] BSI PAS. "212:2016 Automatic resource discovery for the Internet of Things - Specification". In: *British Standards Institution* (2016).