Haojian Jin - haojian@ucsd.edu

Modular Privacy Flows[1] short summary

A very short summary: Systems today often implement fine-grained data access control using all-or-nothing binary APIs. Future systems may consider offering a small set of stateless data transformation operators instead. Developers can connect these operators into a graph to customize desired data access, save the graph as a text-based manifest, and upload the manifest to declare the data access.

Computing systems often allow developers to access more data than needed, violating the principle of data minimization - **a data controller should limit data collection to only what is necessary to fulfill a specific purpose**. For example, most calendar applications only allow users to grant third parties either full access to all their calendar events or none (Fig. 1), even though third parties often only need a small portion. Conventional wisdom is to ask data controllers to offer many levels of data access permissions for every potential use case (Fig. 2). However, enumerating all granularities of permissions would become onerous for system builders (e.g., Google) to implement, unwieldy for users to configure, and complex for developers (e.g., Zoom developers) to learn. For example, a productivity tracking application, interested in knowing how busy the user is, may want to request the number of daily business meetings. Meeting scheduling tools interested in learning whether a user is available (e.g., Doodle), only need to request the time availability blocks.

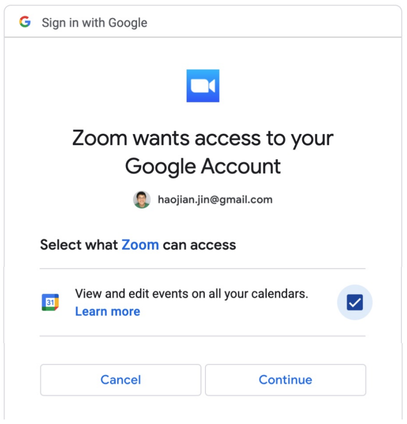
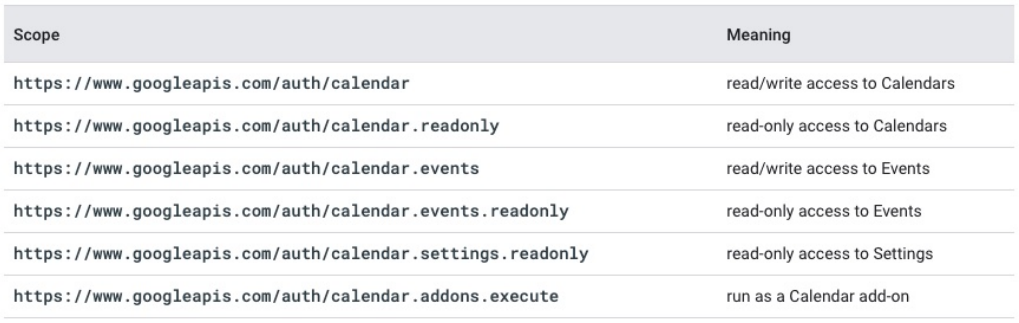

Sign in with Google
Zoom wants access to your Google Account
haojian.jin@gmail.com
Select what Zoom can access
View and edit events on all your calendars.
Learn more
Cancel
Continue

Figure 1: If the user consents, Google would allow Zoom to access all her calendar data.



Scope	Meaning
`https://www.googleapis.com/auth/calendar`	read/write access to Calendars
`https://www.googleapis.com/auth/calendar.readonly`	read-only access to Calendars
`https://www.googleapis.com/auth/calendar.events`	read/write access to Events
`https://www.googleapis.com/auth/calendar.events.readonly`	read-only access to Events
`https://www.googleapis.com/auth/calendar.settings.readonly`	read-only access to Settings
`https://www.googleapis.com/auth/calendar.addons.execute`	run as a Calendar add-on

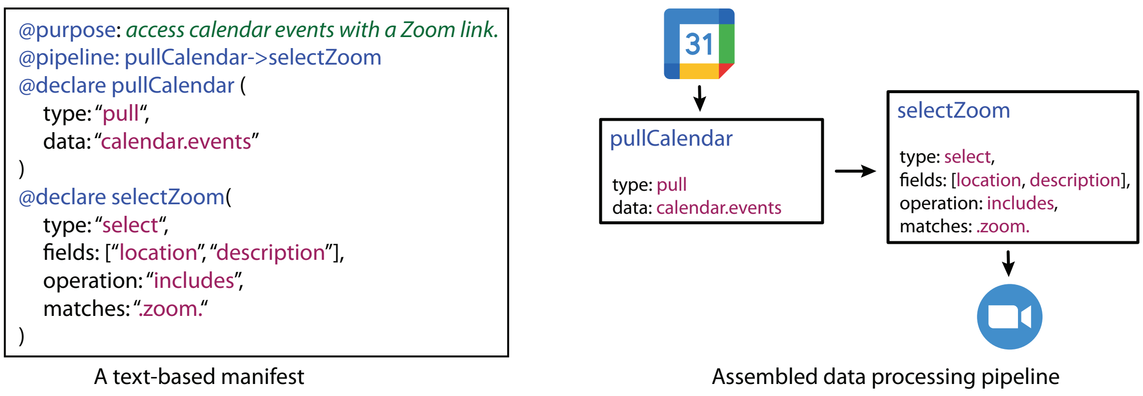
Figure 2: Google calendar offers six types of data access for third parties to access users' calendar data information. The design of read/write/execute permissions is possibly inspired by the Unix file permissions.

This dissertation argues that the conventional design of all-or-nothing binary permissions (Fig. 2), initially designed for security management, has become increasingly insufficient in managing privacy. First, the binary permission approach works best with a small number of options (e.g., Unix file permissions). But, managing privacy requires many fine-grained data access to capture the nuanced data collection/use contexts (e.g., what data is being collected and why, how the data is being used). Second, data analysis can subvert these binary permission systems easily. For example, Zoom can potentially use calendar data to infer many insights about the user, such as income and education. Instead of always offering raw data, systems should seek to process the raw data into less privacy-intrusive forms to reduce these unintentional information leaks.

We introduce a new design pattern, called *Modular Privacy Flows* (MPF). MPF replaces these all-or-nothing permissions with data collection manifests, which have three key ideas. First, developers must declare all intended data collection behaviors in a text-based **manifest** (Fig. 3). Second, to specify the

[1]This document summarizes Haojian's Ph.D. dissertation, "Modular Privacy Flows: A Design Pattern for Data Minimization." It results from multiple-year discussions with Jason Hong, Swarun Kumar, Yuvraj Agarwal, Gram Liu, David Hwang, and many others. This work was funded in part by Cisco, Intel, Infineon, the National Science Foundation, and Air Force Research Laboratory.

data collection, developers choose from a small and fixed set of open-source, chainable, stateless **operators** with well-defined data transformation semantics, authoring a stream-oriented pipeline similar to Unix pipes. Third, a trusted **runtime** enforces the declared behaviors in the manifest, by running all of the pre-loaded, open-source operators specified in the manifest.



```
@purpose: access calendar events with a Zoom link.
@pipeline: pullCalendar->selectZoom
@declare pullCalendar (
    type: "pull",
    data: "calendar.events"
)
@declare selectZoom(
    type: "select",
    fields: ["location", "description"],
    operation: "includes",
    matches: ".zoom."
)
```

Figure 3: Zoom developers can export the data processing pipeline as a text-based manifest (left) and upload it to Google. Given a manifest, Google then assembles and executes a pre-processing pipeline (right) using operators pre-loaded on the cloud, processing users' calendar data before sending it to Zoom.

Combined, developers can reduce data collection by running data transformation tasks before accessing users' data. System builders can implement fewer APIs by reusing a small set of operators. This design also makes the API easier for developers to learn. In addition, since each operator's semantics is known and the manifest is public, outsiders (e.g., regulators and privacy advocates) can quickly analyze developers' data collection behaviors.

Further, MPF can enable many independent end-user privacy features without support from app developers. For example, a future calendar application may allow users to noisify the data provided to developers by inserting additional data transformations into the pipelines specified by developers. Or the calendar application and third-party privacy advocates can analyze the manifest and generate natural language statements to make it easier for non-tech savvy users to understand what data will be sent out, when, and to where. Finally, if services/apps/devices share a similar MPF protocol, we can enable a unified and centralized privacy management interface for end-users.

BROADER APPLICATION DOMAINS. We can potentially apply MPF to many other contexts.

1. **Browser extension**. Browser extensions are much more dangerous than most people realize. Extensions often have access to everything users do online, such as passwords, web browsing histories, and more. Future browsers may consider adopting MPF and require all developers to submit a manifest to access users' data.
2. **Personal data store**. The zoom-google-calendar data access is an example of personal data store. We can also apply MPF to email and wearable health data.
3. **Smart home apps**. Imagine a developer of a smart thermostat claims to only send aggregated temperature history data to their servers once a week. How can outsiders validate this claim, since the hardware, firmware, and backend servers are proprietary black-boxes? Future smart home architectures may leverage an in-home hub to pre-process and minimize outgoing data through MPF. App developers must declare all intended data collection behaviors in a text-based manifest, including under what conditions data will be sent outside of the home to cloud services, where that data is being sent to, and the granularity of the data itself. An in-home trusted hub mediates between all devices in the home and the outside Internet. This hub enforces the declared behaviors in the manifests, and also locally runs all of the operators specified in these manifests to transform raw data before it is relayed to any cloud services.
4. **Social network apps**. The Facebook / Cambridge Analytica data scandal is another data access design fiasco. While the initial app "thisismydigitallife" built by Aleksandr Kogan only had 270 thousand participants, the app accessed the information of each participant's friends, harvesting

data from up to 87 million Facebook users. Future social network apps may adopt MPF in designing data access for third parties.

5. **Mobile apps**. Mobile apps are experiencing an explosion of permissions. Consider the case of requesting mobile location data. There are three relevant permission dimensions: category (background or foreground), accuracy (fine-grained or coarse-grained), and user choices (”while using the app” vs. ”only this time” vs. ”deny”). Further, developers may infer users' location information through network data (e.g., IP address, Wi-Fi MAC address). Applying MPF to mobile apps can introduce two benefits. First, system builders can implement a small set of reusable operators to compose many data permissions. Second, these manifests can enhance the privacy protection of many data resources beyond GPS sensors. For example, instead of always sending developers the raw IP, a manifest may preprocess the IP address into a more privacy-friendly format.
6. **HTTP cookies**. Targeted ads per se does not seem fundamentally evil, unless you think putting car ads in car magazines is also evil. What makes many users uncomfortable is that the ad tech industry tracks users' online activity, inferring many insights that users are unaware of and using these insights in unknown contexts. One alternative solution is that browsers may allow advertisers to access users' attributes through MPF manifests. For example, browsers may introduce a set of operators which can compute users' interests using their local browsing history. Instead of tracking users using unique IDs, advertisers can personalize the ad by querying the attributes computed locally.

FAQ

1. Why would developers adopt data minimization, since developers always want to collect more?
The Fogg Behavior Model shows that three elements must converge at the same moment for a behavior to occur: Motivation, Ability, and a Prompt.

- Many privacy regulations, such as General Data Protection Regulation (GDPR), California Privacy Rights Act (CCPA), and Fair Information Practice Principles (FIPPs), have required developers to implement data minimization. Meanwhile, consumers are looking for products that better respect their privacy.
- Developers today may feel that data minimization is not flexible. For example, at the time of data collection, developers are often unaware of the analysis they want to do in the future. However, this does not mean developers should collect and store all the data for unanticipated future usage. Instead, we should seek new interaction paradigms and development supports to help developers collect data on a need-to-know basis. For example, developers may update the MPF manifests without users' explicit permission. The ecosystem can rely on third-party auditors (e.g., consumer reports, app store, GDPR) to actively search for wrongdoers through automated tools.
- Hopefully, a few developers will start to adopt data minimization. The transparency enabled by MPF can create a virtuous cycle ecosystem where building trustworthy systems is rewarded, and developers compete to guarantee greater user protection, not less.

2. OK. We plan to adopt data minimization, but why should we choose MPF?
Besides the fine-grained permission approach and MPF, there are two potential alternatives: GraphQL and remote code execution. Table. 1 compares MPF with these three approaches. In the database approach, the system may allow developers to author SQL-like queries (e.g., GraphQL) and create a SQL database engine to interpret the declarative queries. MPF has two important advantages over this design: flexibility and auditability. First, it is more flexible for the system builder to extend supported data transformations by adding/removing/updating operator implementations, while modifying a database engine is non-trivial for most organizations. Second, the pipeline approach is more verifiable than the database approach. The runtime simply connects data transformation functions into a pipeline. Outsiders can verify the runtime by verifying each operator's pre-loaded implementation (open-sourced). In contrast, the complexity of a database engine makes it hard to analyze and verify the engine's behavior.

Another alternative design is that system builders may allow developers to upload code to transform data. While this approach offers great flexibility for adding new data accesses, it is hard to enforce (i.e., analyze and control) the data transformation behaviors of these arbitrary programs. In contrast, MPF only allows developers to specify data collection behaviors using a set of high-level operators with well-defined semantics. So MPF can easily infer the program behaviors by analyzing the manifest and help users control how their data would be collected by stopping undesired data flows.

Table 1: Tradeoffs of different data minimization implementations

	Fine-grained permission	MPF	Database (e.g., GraphQL)	Remote code execution
Flexible granularity	✓	✓	x	✓
Easy development	x	✓	✓	x
Enforceable collection	✓	✓	✓	x
Auditability	✓	✓	x	x

3. How can a small set of operators represent numerous fine-grained APIs?
Each MPF operator is like an abstract class in Object-oriented programming, where each operator's behaviors are determined by its properties. Further, all operator abstractions are verbs (e.g., select, pull, detect), mapping to one type of stateless data transformation. For example, depending on the property configurations, a select operator may select a row from a table, a face from an image, and speech from an audio file. Although there might be multiple property options of the same operator, each operator's input and output semantics would be consistent. Using these operators to organize these implementations can make it much easier to analyze the program behaviors.

Given the property specification in the manifest, the runtime then maps the operator to the corresponding implementation and assembles the data transformation executable. Note that we do not need to enumerate all potential data transformation implementations. A small set of simple and standard data processing algorithms can significantly improve privacy protection. For example, simply aggregating tabular data into a scalar number can eliminate many unnecessary privacy risks.

4. How about users? Do they have to read the manifests? How do they manage these fine-grained permissions?
We do not expect users to read these manifests actively. Many studies find that users do not have the expertise and time to manage even binary permissions. Our goal, instead, is to empower third-party auditors and privacy advocates in the ecosystem and disincentivize wrongdoers.

We expect these manifests to be public, making it possible for first-party data providers (e.g., App Store, Calendar apps) and third-party auditors (e.g., Consumer Reports) to analyze manifests programmatically at scale. Users can also see if the required data granularity make sense, and flag items in a review if they do not, block certain outgoing data, or choose not to install an app. Users do not need to interact with these manifests individually. Instead, we expect many end-user-facing features built upon the protocol to help users manage their privacy.

5. Any more questions?
Please contact Haojian Jin (haojian@ucsd.edu)