ing known structures (*e.g.,* JSON, XML, YAML)—an API, by definition, must publicly declare the data format it uses. To share user preferences, collectors must declare in their manifest the type (*e.g.,* location) and granularity (*e.g.,* restaurant name) of the shared data, and the sources used to compute these preferences. Users have control over what data is shared and whether the submission should be anonymous or not.

Consider a restaurant recommendation collector. It continuously monitors a user's location, performs a reverse geocode on the GPS co-ordinates using an online service such as Google Places API, gets back a JSON object representing the address corresponding to the location. This JSON object has a known structure—it contains a field that represents the kind of location whose values might be "Restaurant," or "Drug Store." Heimdall requires collectors to declare the data they will disclose to the recommendation system cloud back-end. This declaration is expressed in terms of the structure and contents of the JSON data—the collector will declare that it will expose the JSON object only if the kind of location is "Restaurant." To ensure that the collector does not manipulate the other fields of the JSON object, Heimdall introduces *immutable blobs* as a security primitive. Any return value from a sensitive data source is immutable. Therefore, even if the collector accesses other sensitive data sources when determining what the most popular restaurant for a particular user is, the output value is guaranteed to not contain information from any of these sources due to immutability. Furthermore, Heimdall provides a secure API to expose immutable blobs, and it ensures that the collector adheres to its manifest by inspecting the structure and contents of the blobs.

Therefore, Heimdall enforces strict privacy guarantees on implicit data collection, thus easing user privacy concerns. Furthermore, recommendation systems do not always require user-identifiable data to provide a general consensus. For example, questions such as "what is the most popular sushi bar in this area?" can be answered without knowing the identity of the visitors to each location. Heimdall allows for such data collection by enabling anonymous submission of preferences.

By granting controlled access to user data and device sensors, Heimdall enables recommendation systems to access implicit preferences, removing their sole reliance on explicit user interactions to measure user preferences. Implicit feedback can be used to measure user interest regarding a particular item, thus addressing the data sparsity problem.

Our Contributions:

- We conduct a study of five online systems with recommendation capability (Goodreads, Yelp, Google Play, IMDB, and Youtube) and analyze the level of user engagement in them. Based on our findings, we distill a set of desired features for collectors and users. (Section 2)
- We develop a prototype of Heimdall on the Android platform (Section 3) and implement a sample set of data collectors spanning the smartphone and smart home environments. These collectors represent the versatility of the Heimdall framework. We perform a thorough evaluation of Heimdall from three perspectives:
  - We perform a series of microbenchmarks that evaluate immutable blob generation time, call latency, and storage overhead of Heimdall. Our results show that API calls in Heimdall impose a $2.3ms$ overhead. An immutable blob of size $1KB$ takes less than $0.3ms$ to generate and only imposes $32B$ of storage overhead. (Section 4.1)
  - We implement three collectors using our framework and discuss developer effort, data collected by the collectors using Heimdall, and the privacy guarantees provided by our system. (Section 4.2)
  - We perform a user study of 166 Amazon Mechanical Turkers to examine the privacy benefits of Heimdall. Our results suggest that the purpose-specific data collection of Heimdall (*e.g.,* recording the name of a restaurant vs. recording fine-grained location information) significantly reduces user concerns.(Section 4.3)

Therefore, in designing Heimdall, we achieve a balance between a user's need for privacy and the need for richer and better recommendations.

2. RECOMMENDATION SYSTEMS: CHALLENGES & OPPORTUNITIES

Online recommendations play a large role in consumer decisions and selections. The number of consumers who search for products online has risen from 41% in 2010 to 50% in 2012 and this number is only expected to increase [30]. With the increased dependence of users on online reviews, services such as Yelp, IMDB, TripAdvisor, and Goodreads aggregate user reviews of various items to produce a general consensus. The reviews that products or locations receive greatly influence their sales and number of user visits—a one-star increase in Yelp reviews of a restaurant increases revenues for that restaurant by $5 - 9\%$ [27].

Although online recommendations and reviews have a large influence over consumers decisions, recommendation systems still suffer posses numerous shortcomings:

- ***Data Collection, Data Breach, and User Privacy:*** Large amounts of data, or big data, provides significant competitive advantages to companies [22] and are correlated with a company's growth and productivity [41]. This is especially true for recommendation systems whose business relies on analyzing user feedback to provide better recommendations and personalization. One major obstacle in collecting and analyzing mass user data is the privacy concerns of users who are worried about abuse of their data and potential tracking of their activities.

  There are multiple dimensions to these users concerns. Data collection in applications happens opaquely with minimal communication to the user about the kind and extent of data [10]. Applications often excessively collect data that is unnecessary for their function [2, 19]. Furthermore, such data collection creates a gold mine of user-identifiable data that is an attractive target for hackers and state-level attackers who regularly steal such confidential information [20]. The Identity Theft Resource Center reported 783 breaches in 2014, which exposed $85,611,528$ user records [12].

- ***Sparsity of Explicit Feedback:*** User feedback is often too few to accurately produce a consensus that reflects the general user opinion. This is especially true for items that are not in the digital or electronics categories [30]. We studied 5 different online recommendation services (Table 1) and compared the number of products (*e.g.,* books, restaurants, movies) to the number of user ratings (*i.e.,* rating bar value) received (Figure 1). We observe that a majority of products do not have even a single user rating, and the products that have ratings contain too few to produce a representative characterization of user opinion. For instance, 70% of products

Source	Type	# of Samples	Definition of Interaction
Goodreads	Books	500	Rating (1-5 stars)
Yelp	Restaurants	10,000	Rating (1-5 stars)
Google Play	Applications	10,000	Rating (1-5 stars)
IMDB	Movies	500	Rating (0.5-5 stars)
Youtube	Videos	500	Up/down vote

Table 1: Data used for evaluating user interaction with different services. Samples were collected at random from each service.

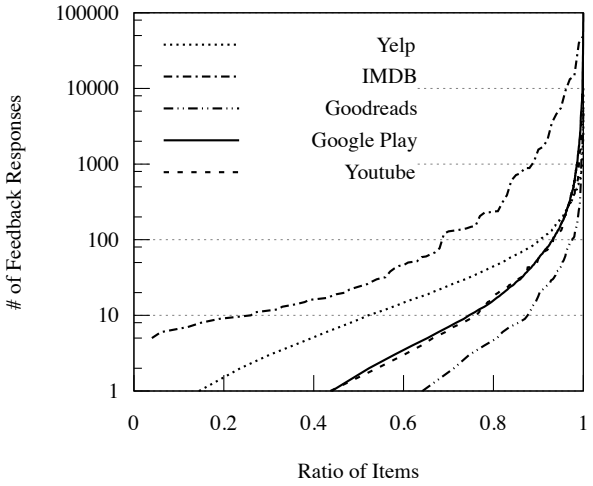

Figure 1: Explicit user feedback (rating bar values) is sparse in general for products from 5 different recommendation systems.

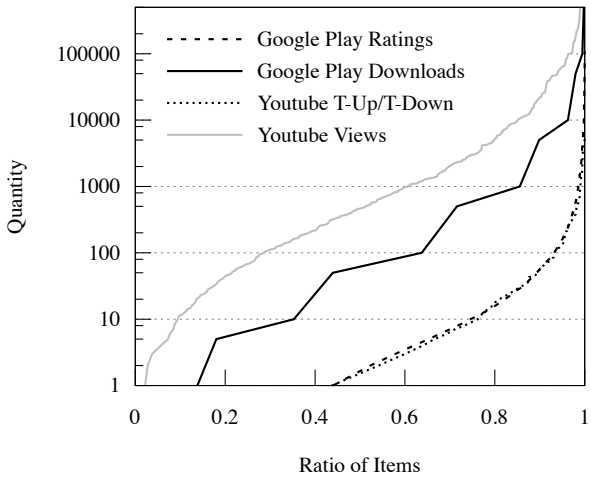

Figure 2: In both Youtube and Google Play store, more than 70% of items have less than 10 reviews. This data suggests that while utilization of products is high, the feedback is still sparse. The Y-axis is interpreted based on the line under consideration. For example, if we consider Google Play ratings, then the Y-axis is the number of ratings, and if we consider Google Play downloads, the the Y-axis is number of downloads. We note that the Google Play ratings and YouTube T-Up/Down lines are slightly different, even though they look visually the same.

on Google Play have fewer than 10 ratings. Recommendation systems recognize this problem, and encourage users to provide more explicit feedback using incentives such as providing discount codes and points-based rewards to those who elect to provide feedback. These techniques have been shown to have limited positive effect [21].

- ***Discrepancy between utilization and feedback:*** One of the reasons for sparse explicit user feedback in the data shown in Figure 1 might be because consumers are not using the majority of products we studied. To confirm that this is not the case, we also measured consumer utilization using approximate metrics appropriate to the type of product. For example, an approximate utilization metric for software on Google Play is the number of downloads,[1] and an approximate metric for video consumption on Youtube is the number of views. Figure 2 shows a CDF for these two utilization metrics for $10,000$ randomly selected apps on Google Play and 500 randomly selected videos on Youtube. We observe that utilization is high but feedback is still low, validating our earlier observation that explicit user feedback fundamentally leads to sparse data. Most of the applications have orders of magnitude fewer reviews than downloads. For applications downloaded at least 1000 times, more than 55% of them have fewer than 100 reviews. Figure 2 also shows the potential to counter data sparsity using implicit feedback. Heimdall unlocks this potential using privacy-respecting implicit feedback collection.

[1]Google Play only provides a range for number of downloads and does not provide exact numbers. In our analysis we conservatively consider the minimum number of downloads.

3. HEIMDALL DESIGN

Heimdall is a framework that enables recommendation systems to build collectors that execute on a user's smartphone or smart home hub. A collector records implicit feedback by observing user activity. Heimdall provides strong privacy guarantees on the type and granularity of data that a collector exposes to the recommendation service cloud back-end, and provides users control over this process. Heimdall directly addresses the fundamental challenge discussed in §2—feedback sparsity, by augmenting explicit feedback with implicit feedback. We design Heimdall to provide these guarantees under the following threat model:

Threat Model. We assume that the Heimdall framework and its underlying operating system are not exploited. This is not a strong assumption as an adversary who has compromised the operating system can freely access data and sensor readings and transfer them regardless of Heimdall. We assume that collectors are untrusted and do not place any restrictions on the kinds of algorithms they run to extract user preferences—they are free to access data existing on the device, from sensors in the built environment, or from knowledge-bases such as Google Maps. Our goal is to prevent collectors from leaking any data that has not been previously approved by the user. We consider addressing side- and covert-channels to be out of scope of the current paper, but we discuss approaches to mitigate them in §5.

The information a collector submits to its recommendation system could implicitly encode additional characteristics. For example, sending a restaurant name implicitly means that the exact GPS co-ordinates of the user are sent out. However, we observe that a GPS co-ordinate does not add information about the activities of the user that cannot be gleaned from the restaurant name itself. That is, it does not add new information. In some cases, new information

can be added. For example, if a collector transmits power consumption of devices, then an attacker could use a model of typical consumption of various devices to learn about the kind of device that generated the power data. In a home setting, this means that an attacker can learn whether a user has a specific type of washing machine (or more generally, a specific kind of appliance). However, such home device information is arguably less privacy sensitive than the actual power consumption data. Nevertheless, we envision that k-anonymity [39] techniques could be adapted to further reduce the privacy loss with such kind of implicit information leakage.

Heimdall does not provide any guarantees on whether the recommendation system (back-end of the collectors) leaks information to third-parties. We leave this end-to-end privacy problem to future work.

3.1 Heimdall Programming Model

Consider the example data collector described in §1 that uses GPS co-ordinates and the Google Places API to infer the restaurants a user visits, and then sends that information to its cloud back-end. The collector requires Internet access to function. Under current smartphone or smart home permission models, use of the collector is tied to an inherent risk of the collector leaking fine-grained location data for tracking a user's movements. Heimdall, however, allows a user to use the collect without such risk.

Listing 1 presents pseudo-code for a minimalistic restaurant collector and Listing 2 presents the associated manifest. Line 2 calls a secure location API that returns the location data as an *immutable blob* (IB). An immutable blob can only be created as the return value from a secure API that Heimdall provides. Once created, it cannot be modified. At Line 3, the collector passes the location immutable blob to the Google Places API that only accepts immutable blob types as arguments. This call returns another IB that represents a reverse geocode of the raw location data. The collector checks if the blob is of the restaurant type, and submits the name value of the blob to the collector's cloud back-end.

Whenever a collector attempts to submit a value, Heimdall will first verify that the value being submitted is a valid immutable blob, and then it will ensure that the submission adheres to the collector's manifest (Listing 2). The manifest provides the user fine-grained control over what data is read and submitted by the collector. In our example, the manifest is expressed in terms of the structure of the return value of the Google Places API. That is, Heimdall leverages the known structure of data objects to support fine-grained policies. Our design formats immutable blobs as JSON strings because they are widely used in RESTful webservices, and are easy to parse. Listing 3 shows example immutable blobs used in our restaurant collector.

We note that the goal of our work is to contribute a mechanism that requires the recommendation collectors to declare the kind of data they release to the recommendation system. In our implementation, this choice is presented to the user during the collector's installation. An interesting question is determining how these choices should be presented to the user and the granularity of the manifest prompt—should we present an all-or-nothing choice to the user while installing a collector, or should we provide more fine-grained control over the kind of data that is released? We envision a user study to determine the appropriate granularity of control, and we leave this to future work.

Collectors run in sandboxes that Heimdall provides. The sandboxes precisely control the data that flows into and out of the collectors within. Whenever a collector tries to submit data to its cloud back-end, it must use a secure API that Heimdall provides that automatically checks the manifest of the collector. Our example collector is obviously very basic, but our goal is to simply provide a high-level picture of the abilities of Heimdall. A real collector may use various other data (*e.g.,* time of the day, past visits, credit card charges) to create an arbitrarily complex model to understand user activity.

Listing 1 Pseudocode for a simple restaurant visit collector. The application collects user's location; uses Google Places API to get data on the location and submits it if it is of type restaurant. A more advanced collector can use a more complex combination of signals to reduce false reports.

```
1  def visitCollector:
2      IB loc = location_service.getloc()
3      IB p = googleapis.place.search(loc[lat],loc[lng])
4      for result in p[results]:
5          if result[types]=="restaurant":
6              submit result[name]
```

Listing 2 Example data collector manifest. The collector uses GPS location and Google geocoding API to collect name of locations of type location.

```
#Sources:
    FINE_LOCATION
    googleapis:place

#Collects:
    @anonymized
    if(googleapis:place:results:types=="restaurant"):
        submit(googleapis:place:results:name,"Link")
```

A collector may set the @anonymized decorator in its manifest. This activates anonymous submissions so that the recommendation system cloud back-end cannot associate the submitted data with any single user's identity. Unless marked for anonymous submission in the manifest, submitted data is transmitted to the collectors' cloud back-end along with the device advertising ID [1]. However, there are many cases where the user is uncomfortable with tying their identity to the submitted data. In some cases, data might be of a sensitive nature (*e.g.,* health related) or users themselves may have higher privacy expectations. In these cases, anonymous submission of data can still provide useful aggregate information on a population's preferences. For example, a collector can reliably extract which physician in your area is more popular with anonymous visit data.

The key primitive providing the desired privacy guarantees in Heimdall is the immutable blob. Listing 4 presents four examples of how Heimdall prevents a collector from accessing or submitting data outside its pre-defined limits:

- The collector is denied access to a resource that has not been requested in its manifest (Line 3-4).
- The collector cannot submit a value that has not been approved in its manifest (Line 8-9).
- The collector cannot pass a parameter that is not of type IB (Line 11-12). All parameters are either an IB or come from a list of static values pre-defined in the API.
- The collector cannot submit a value that does not meet the conditions defined in its manifest (Line 18-19).

These qualities ensure that data entering and leaving a collector always meet the strong type-checking defined in the collector's manifest.

Listing 3 Structure of location and Google Places immutable blobs. Heimdall provides access to sensor and user data (*e.g.,* location, contacts). Web services RESTful APIs naturally produce JSON structures that are stable and well documented. Heimdall defines standard libraries for trusted services (*e.g.,* Google Maps) on top of these APIs.

```
"Location"
{
    "lat": 40.7155809802915,
    "lng": -73.9599399197085,
    "time": "2012-04-23T18:25:43.511Z"
}

"googleapis:place"
{
    "results":[
        { "name":"Subway",
          "types":["restaurant"],
          ... }
        ...
    ]
}
```

Listing 4 Sample of possible malicious activities a collector may attempt and how they are blocked in Heimdall.

```
1  def malicious visitCollector:
2
3      IB call_history = phone_service.getcalls()
4      #Error: Access to phone_service is denied.
5
6      IB loc = location_service.getloc()
7
8      submit loc
9      #Error: Location is not one of the approved data
   ↪    types for collection.
10
11     IB p = googleapis.place.search(38.8977,77.0365)
12     #Error: Parameters are not of type
   ↪    IB:Location:lat and IB:Location:lng
13
14     IB p = googleapis.place.search(loc[lat],loc[lng])
15
16     for result in p[results]:
17         if result[types]=="doctor":
18             submit result[name]
19     #Error: googleapis:place:results:name does not
   ↪    meet the required conditions for collection.
20
```

3.2 Heimdall Framework

As discussed above, collectors enable recommendation systems to record implicit user preferences in a privacy-respecting way. From a framework perspective, Heimdall uses two mechanisms that enable useful and secure collector construction (Figure 3): (1) Sandboxes in which collectors execute; (2) a trusted service that makes sensitive resources available to sandboxed collectors using immutable blobs. A collector in a sandbox can only communicate with the trusted service. Furthermore, the trusted service ensures that only immutable blobs cross sandbox boundaries.

We discuss the design of these components in the context of the Android OS. We choose Android because the availability of source code and ease of access to a wide variety of user data. Basing our design on Android also allows Heimdall to be applicable to smart home environments through Android variants such as Brillo [3] and Google Weave [4].

Collectors and Immutable Blobs. Developers write collectors for Heimdall in Java. A collector runs inside a sandbox that controls its communication with the outside world. A collector may only access sensors, user data, and communication methods (*e.g.,* methods to transmit data to the cloud) using a secure API that Heimdall provides. Furthermore, these secure API calls only use immutable blobs. Each immutable blob is a tuple: $< D, P, Sig >$, where D is a JSON structure containing sensitive data, P is a path inside D and is used when the collector wants to refer to a specific item in D, and Sig is a SHA256 HMAC of D generated by the trusted service. Any parameters used to invoke API calls by the collector are in form of immutable blobs. The trusted service verifies the Sig whenever the collector attempts to transmit an immutable blob outside the sandbox boundaries (*e.g.,* either for submission to the cloud back-end or as parameters to call other APIs). This ensures that any data leaving the sandbox is known and verified by the trusted service.

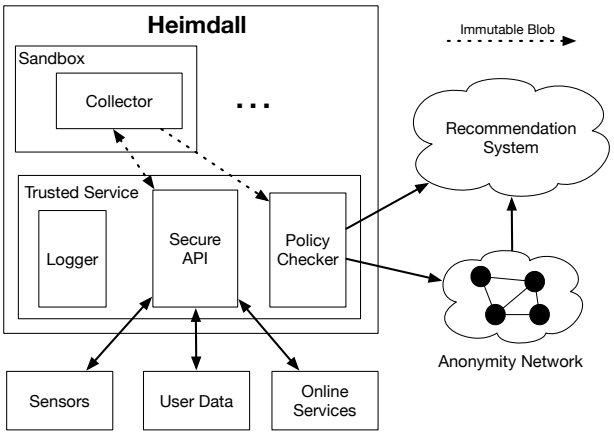

Figure 3: Design overview of Heimdall. Collector can gain access to sensor, user data, and online services through Heimdall secure API. Submission to recommendation system can happen in two ways: 1) Direct submission which includes the device advertisement ID, or 2) Anonymous submission which is sent through an anonymity network (*i.e.,* TOR)

Secure API. The secure API enables collectors to access sensitive data. The API falls into two categories:

- ***Local API:*** provides access to local data (*e.g.,* browsing history, contacts), sensors (*e.g.,* gps, camera), and devices for smart home environments (*e.g.,* door lock, switch) . To keep the structures of immutable blobs uniform, the secure API converts data into pre-defined JSON structures similar to the structure presented for location in Listing 3.

- ***Online Services API:*** provides access to selected online services that Heimdall trusts (*e.g.,* Google Places, Weather Channel). A collector cannot query these services with arbitrary parameters. The parameters used to call the online services can only come from immutable blobs that have been obtained from local sensors or queries previously run on these online services. As these systems use REST APIs with parameters whose structure is known, they automatically supply a JSON response that can be signed and transferred as an immutable blob by the trusted service to the collector.

Policy Checker. A collector can only submit data to its corresponding online recommendation service through a submit function provided by the trusted service. On submission of data, the policy checker will compare the submitted immutable blob

against the approved list of submissions in the collector's manifest and verify that it meets the required constraints. In Listing 2, on submission of an immutable blob, the policy checker first verifies the signature Sig. Next it verifies that the path P inside the blob matches the legal submission types defined in the manifest (*i.e.,* `googleapis:place:result:name` and the JSON structure meets the conditions defined in the manifest (*i.e.,* `googleapis:place:results:types =="restaurant"`). Our implementation at this time supports relation constraints (*e.g.,* IF, AND, OR). If these conditions are met, the policy checker will transmit the data to the submission link that is pre-defined in the collector's manifest. If the anonymization flag is active, the transmission will be conducted through the TOR network to anonymize the sender's identity. Otherwise, the submission will include a unique device identifier associated with the device running Heimdall. Any other attempt to leak an identifier has to be declared in the collector's manifest and will be detectable to the user.

Logger. Heimdall provides transparency about the activities of collectors by providing a detailed log of every interaction a collector has with the trusted service. Logging dissuades malicious collectors from using side-channels as a signaling method and also enables auditing of collectors. Future work could also provide a potential platform for anomaly detection to impede side-channels. Creating such a detector, however, is outside of scope of this work and we leave it for future work.

4. IMPLEMENTATION AND EVALUATION

We implemented Heimdall on a smartphone running Android 4.1 and performed the evaluations on a Nexus 4 device. Heimdall creates the sandboxes with the isolatedProcess flag set. Setting the isolatedProcess flag causes Android to activate a combination of restrictive user IDs, IPC limitations, and strict SELinux policies. These restrictions prevent isolated processes from communicating with the outside world, except via an IPC interface connected to the Trusted Service. The Trusted Service provides an API through which collectors can access sensors and data on the device. It also generates and validates immutable blob signatures on API calls or submission. To give Heimdall access to smart switches, we developed a simple application on the Samsung SmartThings platform that reports the status of all smart switches connected to the SmartThings hub. We use this app as a source in our framework.

We evaluate Heimdall from multiple perspectives. First, we run a series of microbenchmarks to study the blob generation and storage overhead, and the latency for calling a secure API. We find that Heimdall adds modest overhead: A blob of size $1K$ bytes takes $282\mu s$ to generate and takes an additional 256 bits to store. A parameter-less API call by a collector has $2.3ms$ overhead on average.

Second, we developed three simple collectors and integrated them with Heimdall. For each collector, we evaluated the data collection capability, the privacy guarantees, and the development effort of the app. Our results show that Heimdall allows recommendation systems to collect the implicit data required and provides privacy guarantees regarding the data collected. Our collectors are on average 194 lines of code.

Finally, we conducted a user study with 166 participants on Amazon Mechanical Turk.[2] We evaluated the effect of Heimdall on decreasing the privacy concerns of users for a restaurant recommendation system. Our results suggest that use of Heimdall significantly lowers users' concerns on sharing their location and browser history data.

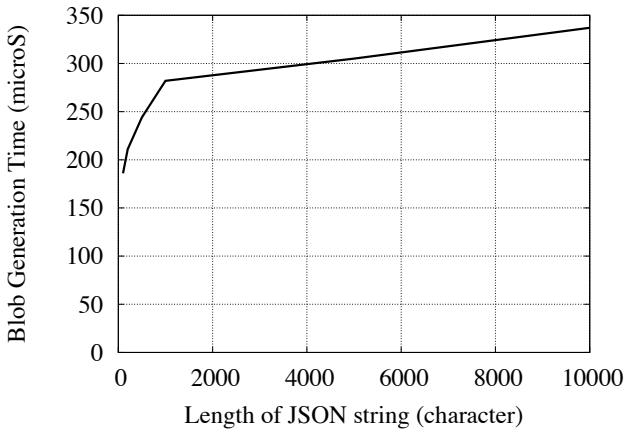

Figure 4: Time overhead of blob generation based on the JSON string size.

4.1 Microbenchmarks

Blob Generation Time. We recorded and averaged 10000 blob generations with data size varying from 100 to 10000 bytes in length. We note that a Google Places API response is $\sim 5KB$ long so our experiments should cover most API response sizes. Figure 4 presents the results of this experiment. The blob generation time increases as the size of the JSON string increases. Note that even when the blob contains $10KB$ bytes, the generation time does not exceed $340\mu s$. This delay does not have a significant negative effect on a collector's functionality as the IPC mechanism used in the operating system has an order of magnitude more latency.

Storage Overhead. A Heimdall collector does not impose a significant memory overhead compared to a collector written outside of the framework. A Heimdall collector only needs to store additional HMACs received for any data that will be submitted to the recommendation system. A SHA256 HMAC is $32B$ in length. As storage/memory space is plentiful in modern phones, this is a very modest overhead.

Call Latency Overhead. We varied the number of parameters of a secure API from 0 to 10. We used immutable blobs with 1000 bytes data length as our parameters and repeated the experiment 100 times for each data point. We observed that as the number of parameters for the secure API increased, the latency increased from $2.3ms$ to $4.2ms$. These measurements include additional latency caused by the IPC and immutable blob verification. This latency is comparable to standard IPC call and is acceptable for collectors' functions.

4.2 Macrobenchmarks

We developed three collector applications for Heimdall framework to evaluate the impact on their capability to collect implicit data from user activities, their privacy guarantees, and the required developer effort. Table 2 presents an overview of these applications. The RestaurantCollector can be used to augment restaurant recommendation services such as Yelp. It tracks and analyses user location and reports the name of the restaurant user have visited to the recommendation system. Using this data, restaurant recommen-

[2]We received exemption from our institution's IRB under title 45 CFR 46.101.(b). Survey data was collected anonymously and did not include any user-identifiable or sensitive information about the participants.

Name / Category	Description	Privacy Risk	Implicit Signal	LoC
RestaurantCollector / Smartphone	Monitors location of smartphone device, infers when the device is at a restaurant, submits name of restaurant to cloud back-end. This helps generate restaurant recommendations.	Can track user's location.	Fine-Grained GPS	359
NewsCollector / Smartphone	Reads user browsing history, detects web pages belonging to news organizations, transmits news items to cloud back-end. This help generate list of most popular news articles.	Sensitive browser history can be leaked to attackers.	Browser History	64
EnergySaver / Smart Home	Monitors power status of devices around the home. Transmits usage of only certain types of high-wattage devices (*e.g.,* oven) if they are used at peak-hour. This helps generate energy saving recommendations.	Can track activity in the home (also, can detect whether the home is occupied or not).	Power state of physical devices	157

Table 2: Example Collectors built using Heimdall.

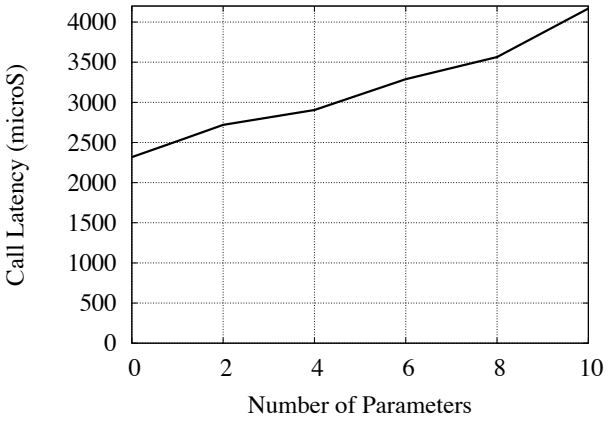

Figure 5: Secure API call latency for different number of parameters. Each Parameter is an immutable blob with $1KB$ JSON data length.

dation system can gain insight into the popularity of each restaurant and how busy they are during different times of day. This application can also potentially use other data such as browsing history (*e.g.,* which restaurant website the user viewed) and purchase history (*e.g.,* how much money the user spent in a restaurant) as additional signals.

NewsCollector can be used by news aggregators such as Digg, Reddit, or feedly. It tracks the browsing history of the user and sends the urls associated with news articles to the recommendation system. Using this collector allows news aggregators to gain insight into popularity of each article in a privacy-respecting manner. This application can be augmented with data such as how much time is spent on each article and subscriptions user has to newsletters for more insight.

Finally, the EnergySaver collector gains access to the status of smart switches in a smart home environment to understand the pattern of energy usage during the day. An application can potentially use this data to provide the user with energy saving suggestions. These three collectors present a diverse set of data that can be gained by relying on implicit user feedbacks.

Data Collection & Privacy. We discuss the data privacy risk that each of the three collectors pose when running on existing platform and examine how Heimdall eliminates those risks successfully.

- *RestaurantCollector:* The collector can potentially track all user movement and misuse this data. Using Heimdall, the collector can still use the location data to extract the restaurants user has visited, but it can only transmit their name to the recommendation system.
- *NewsCollector:* This collector can leak all user browsing history to an attacker. Using Heimdall, the collector is limited to reporting the urls from news organization domains. It cannot report any other urls in an attempt to leak the user's browsing history.
- *EnergySaver:* It has the potential to leak data on all devices usage in a smart home environment. Using Heimdall, the collector can only report certain types of high-wattage devices if they are being used at peak hours.

Developer Effort. Developing collectors for Heimdall does not impose a significant burden over implementing the collector as a regular app in the Android environment. Table 2 provides the number of lines of codes it took to write each collector. When developing a collector in Heimdall, the developer is limited to using Heimdall provided API for accessing user data. These APIs return their results in form of immutable blobs. Much of the extra code developers is required to develop is to maintain and use the immutable blob structure. We envision that with appropriate tool support, many boiler plate tasks for maintaining these blobs can be automated.

4.3 User Study

To evaluate whether Heimdall is a step towards reducing user privacy concerns, we conducted a user study with 166 Amazon Mechanical Turk users. We asked the participants how comfortable they are about a restaurant recommendation service accessing various private data and asked them to rate their level of concern (Likert Scale). We provided the participants with the following description and choices:

> *We are building a restaurant recommendation app for Android and iOS. For it to provide good recommendations, we need to access various kinds of information*

Implicit Signal	Question	Avg. Level of Concern (1-5 Range)
Fine-Grained GPS	[Q1] The recommendation app records and transmits your exact location all the time to its cloud service	3.31
	[Q2] The recommendation app records and transmits your exact location to its cloud service only when you are using the app	2.68
	[Q3] The recommendation app records and transmits only the name of the restaurants you visit to its cloud service	2.41
Purchase History	[Q4] The recommendation app records and transmits your purchase history to its cloud service	3.15
	[Q5] The recommendation app records and transmits how much you spend on meals to its cloud service	3.03
Browser History	[Q6] The recommendation app records and transmits your browsing history to its cloud service	3.42
	[Q7] The recommendation app records and transmits only restaurant websites you visit using your browser to its cloud service	2.61

Table 3: The seven questions in our user study of 166 MTurk participants. All questions concern a restaurant recommendation collector. After filtering for random clickers, we averaged the level of concern users reported for 150 participants. The level of concern is encoded as 1 = not concerned, 5 = would not allow this data collection to happen. We see a significant reduction in concern between Q1 and Q3, and between Q6 and Q7. We did not see a significant reduction in concern between Q4 and Q5, partly due to the general sensitivity of purchase history. Q3, Q5, and Q7 represent the behavior of a Heimdall collector.

about you from your smartphone. In what follows, we will describe type of information being collected. This collected information is transmitted to our recommendation system cloud service. Your task is to rate your feelings towards such collection based on the following scale:

1. ***Not Concerned*** *- Select this option if you are not concerned about sharing the specified data with third parties.*
2. ***Mildly Concerned*** *- Select this option if you are mildly concerned about sharing the specified data with third parties.*
3. ***Concerned*** *- Select this option if you are concerned about sharing the specified data with third parties.*
4. ***Very Concerned*** *- Select this option if you are very concerned about sharing the specified data with third parties.*
5. ***Would Not Allow*** *- Select this option if you would not allow the specified data to be shared with third parties.*

After filtering for random clickers with the help of responses to trap questions [29], we were left with 150 filtered responses. Table 3 contains the survey questions and the average level of concern that users reported. Q1, Q2, Q4, and Q5 represent the behavior of current collectors. Q3, Q5 and Q7 represent the behavior of Heimdall collectors. We conducted a paired t-test for Q1 and Q3 (see Table 3) and found a significantly reduced level of concern ($p < 0.05$) when the recommendation app only collected fine-grained and relevant information (*i.e.,* a location value only when it is a restaurant). A t-test for Q2 and Q3 indicated a significant effect ($p < 0.05$)—even if data is being collected only when the app is being used, users are significantly less concerned when Heimdall informs them that only restaurant names are sent out.

We conducted another paired t-test for Q4 and Q5, but did not find a significant effect ($p > 0.05$). We believe the reason for no significant reduction in level of concern is probably because of the generally sensitive nature of purchase data—be it specific to a kind of item or generic purchase data. Finally, we conducted a paired t-test for Q6 and Q7 and observed a significant effect ($p < 0.05$) suggesting a reduced level of concern. As Q3, Q5, and Q7 represent the behavior of Heimdall collectors, these results suggest that users are possibly more comfortable with purpose-specific data collection that Heimdall enables as compared to the current state of the art where no restrictions exist on how and what data recommendation systems can implicitly record. Although this brief user survey suggests that Heimdall is a step in the right direction, our results do have limitations.

User Study Limitations. Our survey questions were presented in the same specific order to all participants—questions relating to how collectors function today followed by questions relating to how collectors function in Heimdall. This ordering is a potential source of option bias.

Even though Heimdall collectors do not arbitrarily collect sensitive information, our study suggests that users are still mildly concerned about data collection. We believe that this level of concern can be further reduced by coupling Heimdall with a mechanism that enforces data-use policies on the recommendation system backend, to gain an end-to-end privacy guarantee that collected data is only used for recommendation purposes. We elaborate further on this aspect in §5.

Our survey does not determine whether users would stop giving explicit feedback because they (possibly erroneously) assume that implicit feedback is sufficient. This is an interesting side-effect of privacy-respecting implicit feedback collection, and we envision a future survey that is designed to study this aspect.

5. DISCUSSION

Implicit vs Explicit Feedback. Heimdall does not replace explicit feedback, but rather augments it with implicit feedback. Use of implicit feedback only allows a collector to infer a user's interest and preferences; it does not replace user experiences reflected in textual reviews or explicit ratings in scoring systems. Heimdall, however, gives a voice to the vast majority of users who do not provide any feedback on their experiences when using recommendation systems.

Targeted Explicit Feedback. Currently, recommendation systems solicit explicit feedback on all products and services irrespective of whether these products and services are relevant to users. Using implicit feedback that Heimdall enables, a recommendation system can personalize how it solicits explicit feedback. For example, if a recommendation system learns that a user has visited two restaurants very frequently in the past month, the system can dynamically generate explicit feedback UI for only *those* two restaurants, that are arguably more relevant to a user than all the other restaurants. We believe that this can incentivize users to provide explicit feedback. We leave it to future work to design such a system.

Pluggable Secure APIs. Although our current prototype supports a limited set of secure APIs, we envision a pluggable architecture (*e.g.,* using dynamic code loading) where modules containing new secure APIs can be connected to the basic Heimdall framework. Open source developers can build and vet such plugin-based secure APIs to extend the functionality of the Heimdall framework.

Limitations. Heimdall allows recommendation systems to access a wide variety of user's implicit signals that they can use to infer user preferences, thus improving the quality of recommendations. In Heimdall, collectors can use arbitrarily complex models to extract user preference from their data (*e.g.,* This user likes California sushi roll). Collectors, however, are not designed to create digests of these preferences. Questions such as "How many times the user has eaten out this month?" cannot be answered by a collector directly because of the limitations imposed by immutable blobs, but they can be answered on the recommendation system by collecting data on user's restaurant visits.

The recommendation system itself can possibly misuse data that collectors submit (*e.g.,* leak user behavior to third parties). Our design currently does not provide any guarantees on the behavior of the recommendation system. Information flow control approaches are a promising step in providing end-to-end privacy guarantees [36], including recent work on structuring code to make flows explicit [17]. Nevertheless, Heimdall collectors are a first step towards achieving the end-to-end privacy goal, and is the focus of this work. Another promising direction to obtain end-to-end guarantees is the notion of policy-carrying data [34]. A collector could attach a policy before submitting to the recommendation system. Then, the recommendation system can only access the submitted data if it publicly acknowledges (in an unforgeable way) that it will follow the rules of the associated policy. If it deviates from the policy, then either technical mechanisms or legal instruments could be used to address the privacy breach.

Side/Covert Channels. Heimdall controls the data that collectors gather by enforcing fine-grained data type tracking through immutable blobs that contain multiple data fields (*e.g.,* type of location, name of location, value of sensor). However, it is possible for a malicious collector to transmit information through side and covert channels, such as encoding data within timing, frequency, or pattern of data submissions. While we don't address these challenges, many of the techniques proposed in the literature (*e.g.,* creating a delay before any submission to protect against timing side channels) are applicable to Heimdall [44,45]. Furthermore, providing an accurate log of collector's activities allows for auditing and anomaly detection.

6. RELATED WORK

Recommendation Systems. As discussed in §1, recommendation systems suffer from three shortcomings: (1) users have privacy concerns while sharing preferences [9, 42, 43], (2) users must take extra steps to provide explicit feedback by interacting with review-related UI elements like rating bars, and (3) users often leave feedback only after an extreme experience [8, 26]. Our key insight in Heimdall is that all of these problems can be solved if recommendation systems could collect implicit preferences in a privacy-respecting manner. Therefore, Heimdall is design with this goal in mind. Heimdall enables recommendation systems to construct collectors that *augment* explicit preferences with a user's implicit preferences.

Automatically collecting implicit preferences can increase the privacy concerns of the user if they do not have control of the process or are unaware that such collection occurs. Heimdall forces implicit preference collectors to declare what data is exported to the recommendation system's back-end. Therefore, users are informed that data collection is occurring and can choose to block undesired collection. Zhang *et al.* showed that adding privacy control mechanisms to implicit data input addressed user privacy concerns [43].

There is a body of work discussing privacy attacks on recommendation services. Calandrino *et al.* leverage temporal changes in the public output of a recommendation service to infer information about a specific customer's transactions with only a small amount of auxiliary information [11]. Ramakrishnan *et al.* discuss how information on the existence of customers with eclectic preferences in products, or straddlers, can be used to reveal personal details of other customers with similar preferences [33]. These are attacks on the algorithm the recommendation system uses to produce recommendations. In contrast, our work is concerned with privately leveraging implicit signals to improve the quality of recommendations. Preventing attacks on the recommendation algorithm are orthogonal to our goals.

Implicit Preference Collection. A recent class of systems are emerging that perform entity recognition and task extraction directly on data that appears on smartphone UI. For instance, Fernandes *et al.* built Appstract [18], and Chandramouli *et al.* built Insider [13]. These systems extract structured data from smartphone app UI and associate semantics to that data. Such extraction creates a rich data source that recommendation systems can use for implicit preference collection. However, these systems leave designing proper security controls for third-party access to this data as future work. We believe that Heimdall is a suitable framework that enables recommendation systems to leverage this rich source of semantically labeled data in a privacy-respecting manner.

Information Flow Tracking. JFlow and Jif are security-typed languages where developers must learn a new type-system to gain security benefits [5, 31]. Heimdall draws on ideas from such languages, but it simplifies the type-system to only include a single new security-oriented type—immutable blobs. A collector may only sink specific immutable blobs declared in its manifest.

FlowFence is a general-purpose information flow control system aims at Internet of Things apps [17] that only allows data to flow from a source to a sink if the data policy permits. In contrast, Heimdall does not monitor data flows from a source to a sink. Instead, it simply ensures that data exported at a sink has been declared in a collector's manifest and has been approved by the user. However, a system like FlowFence, which supports information flow control as a first-class primitive (as opposed to bolt-on approaches discussed

below), can be used on the recommendation system itself to obtain end-to-end privacy guarantees that limit the misuse of data submitted by collectors.

A large body of work deals with dynamic and static taint analyses that track how data flow of a program affects the contents of variables containing sensitive information [36]. These techniques, which introduce large performance overheads, are generally applied in non-adversarial settings such as test generation and integrity protection, where the attacker does not control the code that is being taint checked. Within an adversarial setting, attackers can defeat current taint analysis techniques using control flows and concurrency [35]. Heimdall does not use taint tracking to achieve its privacy goals because the application area is very specific and does not require tracking data and control flow—a collector's only function is to sink an immutable blob whose contents, by definition, cannot be changed by data or control flow of the collector's code.

Linkability & K-Anonymity. Heimdall enables users to anonymize their submissions to the recommendation system using an anonymity network such as TOR. While this approach removes any direct link between different data submissions of an individual, it does not provide an unlinkability guarantee. Prior work such as Narayanan *et al.* [32] has shown how large sparse datasets of recommendations can get deanonymized by collusion between different sources. Providing such guarantees fall outside our threat model, but has been extensively studied in K-Anonymity literature [23–25, 28, 38, 40] .

7. CONCLUSION

Many users today rely on online recommendation systems (*e.g.,* Yelp, Goodreads, IMDB) to better inform their decisions about whether or not to use a certain product or service. These systems seek to aggregate large amounts of implicit (*e.g.,* GPS data, purchase history, browsing history) and explicit (*e.g.,* written reviews, ratings) user feedback in order to produce high quality recommendations. However, due to user privacy concerns, these systems are not given access to implicit data sources, and thus must rely on explicit feedback, which most users do not provide. We have proposed Heimdall, a practical privacy-respecting preference collection framework that allows recommendation systems to collect implicit feedback. Implicit preference collectors declare the type (*e.g.,* location) and granularity (*e.g.,* restaurant name) of data they share. Users approve or reject these requests to share data and Heimdall ensures that the collectors behave in accordance with their declared data accesses and the user's decision. Heimdall introduces immutable blobs as a mechanism to help provide these guarantees. We built Heimdall for Android (and also interfaced it with Samsung SmartThings). Our evaluation indicates that performance overhead is minimal—generating immutable blobs adds a $340\mu s$ delay to IPCs that transfer data into a collector. We conducted a user study with 166 MTurk participants to determine whether Heimdall collectors decrease privacy concerns. We find that users are significantly less concerned with data collection when they have control over the collection process and when the nature of the collected data is made explicit (*e.g.,* collecting fine-grained location vs. using location to only collect restaurant names).

Acknowledgements

We thank Chunyi Peng, our shepherd, and the anonymous reviewers for their insightful feedback. This material is based upon work supported by the National Science Foundation under Grant No. 1318722. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

8. REFERENCES

[1] Advertising id. https://support.google.com/googleplay/android-developer/answer/6048248.

[2] Apple Q&A on location data - April 27, 2011. http://www.apple.com/pr/library/2011/04/27Apple-Q-A-on-Location-Data.html.

[3] Google Brillo OS for IoT. https://developers.google.com/brillo/.

[4] Google Weave for IoT. https://developers.google.com/weave/.

[5] JIF. http://www.cs.cornell.edu/jif/.

[6] Quantcast number of visitors ranking. https://www.quantcast.com/top-sites/US/1.

[7] Why Do Consumers Leave Reviews? https://www.getfivestars.com/blog/survey-why-do-consumers-leave-reviews/.

[8] G. Adomavicius and A. Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *Knowledge and Data Engineering, IEEE Transactions on*, 17(6):734–749, 2005.

[9] N. F. Awad and M. S. Krishnan. The personalization privacy paradox: An empirical evaluation of information transparency and the willingness to be profiled online for personalization. *MIS Q.*, 30(1):13–28, Mar. 2006.

[10] J. L. Boyles, A. Smith, and M. Madden. Privacy and data management on mobile devices. *Pew Internet & American Life Project*, 4, 2012.

[11] J. A. Calandrino, A. Kilzer, A. Narayanan, E. W. Felten, and V. Shmatikov. "you might also like: " privacy risks of collaborative filtering. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, SP '11, pages 231–246, Washington, DC, USA, 2011. IEEE Computer Society.

[12] I. T. R. Center. Breach report 2014. 2014.

[13] V. Chandramouli, V. N. Abhijnan Chakraborty, S. Guha, V. Padmanabhan, and R. Ramjee. Insider: Towards breaking down mobile app silos. In *TRIOS Workshop held in conjunction with the SIGOPS SOSP 2015*, September 2015.

[14] E. Chin, A. P. Felt, V. Sekar, and D. Wagner. Measuring user confidence in smartphone security and privacy. In *Proceedings of the Eighth Symposium on Usable Privacy and Security*, page 1. ACM, 2012.

[15] Z. Fang, W. Han, and Y. Li. Permission based android security: Issues and countermeasures. *computers & security*, 43:205–218, 2014.

[16] A. P. Felt, K. Greenwood, and D. Wagner. The effectiveness of application permissions. In *Proceedings of the 2nd USENIX conference on Web application development*, pages 7–7, 2011.

[17] E. Fernandes, J. Paupore, A. Rahmati, D. Simionato, M. Conti, and A. Prakash. Flowfence: Practical data protection for emerging iot application frameworks. In *25th USENIX Security Symposium (USENIX Security 16)*, pages 531–548, Austin, TX, Aug. 2016. USENIX Association.

[18] E. Fernandes, O. Riva, and S. Nath. Appstract: On-the-fly app content semantics with better privacy. In *Proceedings of the 22Nd Annual International Conference on Mobile*

Computing and Networking, MobiCom '16, pages 361–374, New York, NY, USA, 2016. ACM.

[19] N. Geri and Y. Geri. The information age measurement paradox: Collecting too much data. *Informing Science: the International Journal of an Emerging Transdiscipline*, 14:47–59, 2011.

[20] G. Greenwald. *No Place to Hide: Edward Snowden, the NSA, and the U.S. Surveillance State*. Metropolitan Books, May 2014.

[21] J. Hamari, J. Koivisto, and H. Sarsa. Does gamification work?–a literature review of empirical studies on gamification. In *System Sciences (HICSS), 2014 47th Hawaii International Conference on*, pages 3025–3034. IEEE, 2014.

[22] E. Junqué de Fortuny, D. Martens, and F. Provost. Predictive modeling with big data: is bigger really better? *Big Data*, 1(4):215–226, 2013.

[23] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, pages 49–60. ACM, 2005.

[24] K. LeFevre, D. J. DeWitt, and R. Ramakrishnan. Mondrian multidimensional k-anonymity. In *Data Engineering, 2006. ICDE'06. Proceedings of the 22nd International Conference on*, pages 25–25. IEEE, 2006.

[25] N. Li, T. Li, and S. Venkatasubramanian. t-closeness: Privacy beyond k-anonymity and l-diversity. In *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*, pages 106–115. IEEE, 2007.

[26] X. Li and L. M. Hitt. Self-selection and information role of online product reviews. *Information Systems Research*, 2008.

[27] M. Luca. Reviews, reputation, and revenue: The case of yelp. com. *Com (September 16, 2011). Harvard Business School NOM Unit Working Paper*, (12-016), 2011.

[28] A. Machanavajjhala, D. Kifer, J. Gehrke, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 1(1):3, 2007.

[29] W. Mason and S. Suri. Conducting behavioral research on amazon's mechanical turk. *Behavior Research Methods*, 44(1):1–23, 2012.

[30] McKinsey&Company. Telecommunications, media, technology - iconsumers: Life online. 2013.

[31] A. C. Myers. Jflow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '99, pages 228–241, New York, NY, USA, 1999. ACM.

[32] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, pages 111–125. IEEE, 2008.

[33] N. Ramakrishnan, B. J. Keller, B. J. Mirza, A. Y. Grama, and G. Karypis. Privacy risks in recommender systems. *IEEE Internet Computing*, 5(6):54–62, Nov. 2001.

[34] S. Saroiu, A. Wolman, and S. Agarwal. Policy-carrying data: A privacy abstraction for attaching terms of service to mobile data. In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, HotMobile '15, pages 129–134, New York, NY, USA, 2015. ACM.

[35] G. Sarwar, O. Mehani, R. Boreli, and D. Kaafar. On the effectiveness of dynamic taint analysis for protecting against private information leaks on android-based devices. In *SECRYPT 2013, 10th International Conference on Security and Cryptography*, pages 461–467, ReykjÃąvik, Iceland, jul 2013. SciTePress.

[36] E. J. Schwartz, T. Avgerinos, and D. Brumley. All you ever wanted to know about dynamic taint analysis and forward symbolic execution (but might have been afraid to ask). In *Proceedings of the 2010 IEEE Symposium on Security and Privacy*, SP '10, pages 317–331, Washington, DC, USA, 2010. IEEE Computer Society.

[37] M. Sheppard. Smartphone apps, permissions and privacy. In *Office of the Privacy Commissioner of Canada*, 2013.

[38] L. Sweeney. Achieving k-anonymity privacy protection using generalization and suppression. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):571–588, 2002.

[39] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[40] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 10(05):557–570, 2002.

[41] P. Tambe. Big data investment, skills, and firm value. *Management Science*, 60(6):1452–1469, 2014.

[42] E. Toch, Y. Wang, and L. F. Cranor. Personalization and privacy: A survey of privacy risks and remedies in personalization-based systems. *User Modeling and User-Adapted Interaction*, 22(1-2):203–220, Apr. 2012.

[43] B. Zhang, N. Wang, and H. Jin. Privacy concerns in online recommender systems: Influences of control and user data input. pages 159–173. USENIX Association, 9999.

[44] D. Zhang, A. Askarov, and A. C. Myers. Predictive mitigation of timing channels in interactive systems. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 563–574, New York, NY, USA, 2011. ACM.

[45] D. Zhang, A. Askarov, and A. C. Myers. Language-based control and mitigation of timing channels. In *Proceedings of the 33rd ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '12, pages 99–110, New York, NY, USA, 2012. ACM.

[46] H. Zhang, K. E. Nejad, A. Rahmati, and H. V. Madhyastha. Towards comprehensive repositories of opinions. In *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*, HotNets '16, pages 15–21, New York, NY, USA, 2016. ACM.