Ancile: Enhancing Privacy for Ubiquitous Computing with Use-Based Privacy

Eugene Bagdasaryan
Cornell Tech, Cornell University
eugene@cs.cornell.edu

Griffin Berlstein
Vassar College
grberlstein@vassar.edu

Jason Waterman
Vassar College
jawaterman@vassar.edu

Eleanor Birrell
Pomona College
eleanor.birrell@pomona.edu

Nate Foster
Cornell University
jnfoster@cs.cornell.edu

Fred B. Schneider
Cornell University
fbs@cs.cornell.edu

Deborah Estrin
Cornell Tech, Cornell University
destrin@cs.cornell.edu

ABSTRACT

Widespread deployment of Intelligent Infrastructure and the Internet of Things creates vast troves of passively-generated data. These data enable new ubiquitous computing applications—such as location-based services—while posing new privacy threats. In this work, we identify challenges that arise in applying use-based privacy to passively-generated data, and we develop Ancile, a platform that enforces use-based privacy for applications that consume this data. We find that Ancile constitutes a functional, performant platform for deploying privacy-enhancing ubiquitous computing applications.

CCS CONCEPTS

• **Security and privacy** → **Access control**; **Information flow control**; *Pseudonymity, anonymity and untraceability*; • **Information systems** → *Location based services.*

ACM Reference Format:
Eugene Bagdasaryan, Griffin Berlstein, Jason Waterman, Eleanor Birrell, Nate Foster, Fred B. Schneider, and Deborah Estrin. 2019. Ancile: Enhancing Privacy for Ubiquitous Computing with Use-Based Privacy. In *18th Workshop on Privacy in the Electronic Society (WPES'19), November 11, 2019, London, United Kingdom.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3338498.3358642

1 INTRODUCTION

The recent proliferation of sensors has created an environment in which human behaviors are continuously monitored and recorded. For example, fine-grained location data are generated whenever a person carries a mobile phone. These *passively-generated* data—which are generated without explicit action by the data subject, and often without the subject's knowledge or awareness—enable many new applications such as smart buildings that reduce energy consumption by only heating or cooling areas where people are present, health applications that improve fitness by encouraging increased mobility, and productivity applications that suggest ad-hoc meetings when a quorum of a collaborative team is present. As is the case for mobile and social applications, *ubiquitous computing applications*, which consume passively-generated data, are often developed by third parties.

Many types of passively-generated data are particularly sensitive. For example, real-time location information could facilitate stalking or other abuse [65]. Presence at particular locations (e.g., certain medical clinics or clubs) might be correlated with sensitive attributes (e.g., health conditions or demographics) [7]. Even when the individual *data values* are not sensitive, aggregate *traces* of passively-generated data may be sensitive. For example, locations traces can be used to identify shopping, fitness, and eating habits [64]. Such traces have been used to set insurance rates [21] and to identify individual users in large, anonymized databases [31]. To develop a trustworthy platform for ubiquitous computing applications, it is necessary to provide strong privacy guarantees for passively-generated data.

Use-based privacy [9, 12, 13, 42], which re-frames privacy as the prevention of harmful uses, appears well suited to address this problem. Use-based privacy associates data with policies that authorize certain types of data use without permitting unrestricted access to that data. These policies typically describe how restrictions change as data are transformed and as other events occur—i.e., they are *reactive* [9, 32]. For example, a policy might state that a smartphone application developed by an insurance provider may use location data to provide roadside assistance but that aggregate location traces may not be used to set insurance rates.

To date, use-based privacy has been implemented only in contexts where sensitive data are *actively generated*, that is where the data subject is explicitly involved in data generation and collection (e.g., health records [9] or survey data [8]). In those contexts, data processing pipelines are known in advance, and there is limited aggregation of sensitive data values. In this work, we explore how use-based privacy can be extended to support ubiquitous computing applications, which consume passively-generated data. Drawing on

WPES'19, November 11, 2019, London, United Kingdom

ACM ISBN 978-1-4503-6830-8/19/11…$15.00
https://doi.org/10.1145/3338498.3358642

a series of example location-based services, we show that such applications rely on data-processing pipelines that combine data from multiple sources in complex and nuanced ways. Privacy challenges that arise in these settings are discussed in Section 2. Capturing the "right" notion of authorization in these applications requires data- and context-dependent policies, as well as the ability to synthesize new policies for derived values such as collections. While it is possible in principle to encode some of these policies in existing use-based privacy frameworks such as Avenance [9], a better approach is to give them a first-class treatment. So this work extends the Avenance language to meet these challenges. The revised policy language is described in Section 3.

We built Ancile, a system which augments an existing intelligent infrastructure with enforcement mechanisms for use-based privacy. Because data are passively-generated, Ancile provides an interface for principals to authorize data import from a *data provider* and to specify policies to be associated with all data about a data subject received by an application from that data provider. These policies are specified as regular expressions over an "alphabet" of commands that operate on data; a policy will specify how a data value may be used and how any derived values may be used. Both data subjects and policy administrators (e.g., regulatory experts or faculty PIs) may specify policies. On data ingress into Ancile, each data value is associated with a policy formed by intersecting the policies provided by each stakeholder. Ancile implements a reactive mechanism that updates the associated policy when a data value is used and that synthesizes policies for any derived data values. To support extensible development of location-based services by third parties, Ancile offers a library of commands that application developers can use to write programs for handling location data. Ancile executes these programs on behalf of the applications and enforces that the data are only processed in compliance with their associated policies. The implementation of the Ancile system is discussed in Section 4.

We deployed Ancile with campus-wide location service and with Android location services. We evaluated its functionality by implementing four example applications. We evaluated system performance through component benchmarks and system scalability via load testing. This evaluation is described in Section 5.

Our initial findings suggest that Ancile is both expressive and scalable. This suggests that use-based privacy is a promising approach to developing a privacy-enhancing platform for implementing location-based services and other applications that consume passively-generated data.

2 APPLICATIONS

To identify privacy challenges that arise in ubiquitous computing applications, we consider four simple applications. We draw these applications from the domain of location-based services because passive generation of such data is widespread [24, 67], the associated privacy risks are well documented [7, 33, 64], and the challenges are representative of ubiquitous applications more broadly. For each application, we investigate how location data might be processed to support application functionality while restricting its use in accordance with the principle of least privilege.

BookNearMe: This application reserves a meeting room based on a user's current location. It looks up a list of rooms using a calendar service and reserves a nearby, available room. A key privacy goal for a BookNearMe user is to maintain the secrecy of their current, fine-grained location. Since precise location information is not necessary to locate a nearby room, approximate data can be used without significantly degrading the quality of service. (The same observation holds for location-based services that find nearby points of interest, such as restaurants, ATMs, or shops). So a program that returns fuzzed location data to the application (which would then reserve an appropriate room) would enhance privacy without precluding utility. The data processing pipeline for this application is depicted in Figure 1a. A fuzzing function that adds zero noise would not enhance privacy. So a policy should be able to specify that location data may be returned to an application only after it has been perturbed with a specified fuzz factor.

Privacy Challenge 1: To handle parameterized functions, the policy language must support argument-dependent authorizations.

RoamingOfficeHours: This application is designed for TAs or professors who wish to hold regular office hours at irregular locations. It publishes a user's current location if the user is currently on campus and the user's calendar has office hours scheduled for the current time. The primary privacy goal for a RoamingOfficeHours user is to maintain the secrecy of their current location when they are off campus or are not currently holding office hours. This goal can be met if data use is context-dependent, that is, location data is only released if the desired conditions (on campus and during scheduled office hours) are true. This data processing pipeline is shown in Figure 1b.

Privacy Challenge 2: To handle context-dependent policies, the policy language must be able to express authorizations that depend on data and external state.

GroupStudy: This application helps small groups of users (e.g., students or developers) collaborate by enabling impromptu face-to-face meetings. It maintains a list of group members and periodically checks whether a quorum of the group is on-site; if so, it notifies all group members. The primary goal for a GroupStudy user is to keep their location secret by only releasing a single bit of information: whether or not a quorum of the group is currently on-site. This goal can be met if each user's location is used only to determine whether or not the user is on site, and if these Boolean values are employed only to evaluate whether a quorum of the group is present. This data processing pipeline is depicted in Figure 1c. Note the use of a function that evaluates a quorum takes many inputs and produces a single output. Ancile must be able to support such *aggregation* functions.

Privacy Challenge 3: To handle uses that take multiple different data values as inputs, the policy language must be able to authorize aggregation functions including synthesizing derived policies for the values they produce.

LocationPredictor: This application is a machine learning service that predicts the next user location based on that user's location trace over time. This application can be used to implement smart

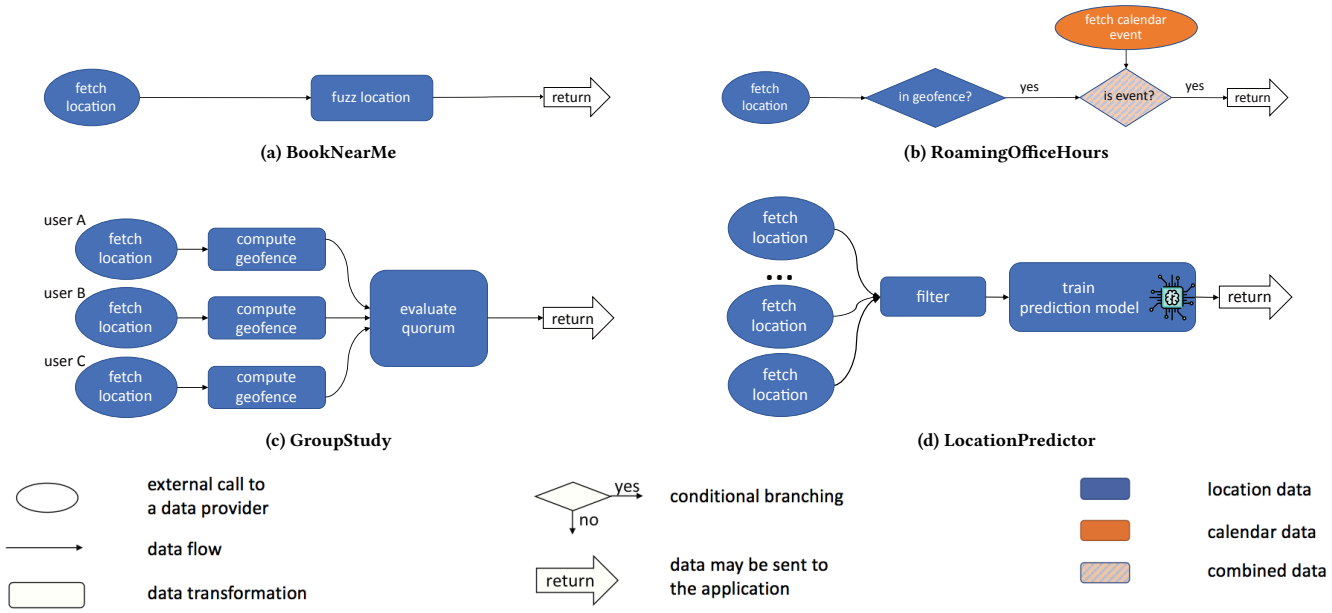

Figure 1: Possible data processing pipelines for privacy-enhancing location-based services.

building management, for example, to forecast high or low density areas and perform temperature adjustment, light adjustment, or elevator positioning. The primary privacy goal for a LocationPredictor user is to prevent location traces from being leaked or being used for any purpose other than training or using the prediction model. Before training the model, the location data must be pre-processed (for example, eliminating data from non-mobile devices). A possible data processing pipeline that achieves this goal is depicted in Figure 1d. Note that it combines many individual location values into a single value (e.g., a list of locations) and then eliminates some of those values. In theory, this could be treated as an aggregation function (construct list) followed by a standard, one-input function (modify the list by filtering out some elements). In practice, however, policy synthesis for aggregate values does not retain provenance information, so it would be difficult to correctly synthesize policies. Consider a user who places a permissive policy for location data collected from their mobile phones (because they want to allow applications to use their location) but a restrictive policy on location data collected from their laptops (to minimize the risk of theft). A data structure containing all location data would be restrictive, but the policy for filtered data (which only contains data from the phone) would be more permissive.

Privacy Challenge 4: To authorize data-processing pipelines that operate on data structures, the Ancile policy language must support functions that create and operate on data structures, including synthesizing derived policies for derived data.

3 POLICY LANGUAGE

Unlike traditional approaches that focus on limiting data collection, use-based privacy [9, 12, 13, 42] expresses restrictions on how data may be used. The approach aligns well with the challenges related to location data, which is both useful and sensitive.

Use-based privacy has been found to need reactive languages for expressing its policies [9]. A reactive language [32] is one in which the current restrictions associated with a value may depend on its history—i.e., how it was derived as well as any environmental events that may have occurred. For example, a policy might prohibit the use of raw location traces—only allowing specified filtering operations—but might authorize the output of those functions to be used without restrictions. Or a policy might only permit release of a user's current location during work hours.

Avenance [9] is a reactive language designed specifically for use-based privacy. In Avenance, current use-authorizations are expressed as triples (I, E, P), where I is an invoking principal (an application), E is an executable (an action that may be performed), and P is a purpose (a reason to get the data). The set of possible use-authorizations forms a finite state automaton; the state of this privacy automaton changes when events (either environmental events or data transformations) occur. Authorization decisions are based on the current state of the privacy automaton.

The Ancile policy language is a variant of Avenance that introduces advanced features that meet the privacy challenges identified in Section 2. An Ancile policy is a regular expression on set of commands C (commands take the place of Avenance executables), as captured by the grammar in Figure 2. For example:

```
encrypt .
  ( (!decrypt)* +
    decrypt . on_campus +
    decrypt . aggregate_trace . compute_home) .
  return_to_app
```

P	$::= C$	–	command
	$\mid P_1 \,.\, P_2$	–	sequential composition
	$\mid (P_1 \;+\; P_2)$	–	union
	$\mid (P_1 \;\&\; P_2)$	–	intersection
	$\mid\, !P$	–	negation
	$\mid P^*$	–	Kleene star
	$\mid 0$	–	no operation

Figure 2: Policy Syntax

This policy would allow encrypted data to be used in any way (`(!decrypt)`*). It would allow plaintext data to be used to determine whether or not the location is on campus; that Boolean value may be exfiltrated from Ancile to an application. It would also allow plaintext data to be aggregated into a location trace, which may be used to infer the data subject's home location; that home location may be exfiltrated from Ancile and sent to an application.

There are two classes of commands:

(1) *Transformations* are commands that take data as input and generate derived data. Ancile policies specify whether a transformation is authorized and, if so, what policy to associate with the derived data.
(2) *Uses* are commands that take a single data value as input and return none. Ancile policies specify whether a use is authorized and, if so, how to modify the policy on the input value when the use occurs.

Ancile has a pre-defined set of transformations $\mathcal{T}$ and uses $\mathcal{U}$. The current implementation supports a variety of transformations that process data in different ways (e.g., `encrypt`, `decrypt`, etc.). The command `return_to_app`, which (as a side-effect) exfiltrates the data value from Ancile to the application, is an example of a use. For convenience, Ancile also allows a policy to use the notation `ANYF` for authorizing any single command.

Ancile policies specify which commands are authorized to take a particular data value (e.g., a location or a location trace) as input. For most commands c, a policy P authorizes c if there exists a string S with prefix c such that $S \in \mathcal{L}(P)$ (where $\mathcal{L}(R)$ denotes the set of strings generated by the regular expression R). A command that sends a data value to the application (e.g., `return_to_app`) is only authorized if the string $S = c \in \mathcal{L}(P)$.

Ancile policies also specify how to synthesize policies for derived values and how to update policies on existing values. Transformations t take an input x and return an output $t(x)$; if P_x is the policy associated with x, then Ancile associates derived value $t(x)$ with a derived policy $D(P_x, t)$, where $D(P_x, t)$ is the derivative [10] of P_x with respect to t. Uses return no values; when an authorized use $u(x)$ occurs, Ancile changes the policy on input x to be the derivative policy $D(P_x, u)$. Intuitively, the derivative policy $D(P_x, c)$ is defined so that a string of commands $S \in \mathcal{L}(D(P_x, c))$ if and only if the string of commands $cS \in \mathcal{L}(P_x)$. A formal definition of how derivative policies are constructed is given in Appendix A.

For example, a policy might state that only anonymized versions of the data may be returned; this policy would be expressed as

```
anon . return_to_app
```

This policy is interpreted as saying that the only command that is authorized for this data is the command `anon` and that the derived value output by this command should be associated with the derived policy $D($`anon . return_to_app`$,$ `anon`$) =$ `return_to_app`.

A slightly more permissive policy might allow either anonymized data or particular simple statistics (e.g., a Boolean value indicating whether a location is within a specified geofence) to be returned to applications; this policy would be expressed as

```
(anon + in_geofence) . return_to_app
```

The derivative policy associated with an anonymized location would be `return_to_app`. Likewise, the derivative policy associated with the Boolean value indicating whether or not this location is inside the geofence would also be `return_to_app`.

Similarly, we can think about a negation operation that permits all the commands except the specified one, for example the following policy would authorize any transformations, but would prohibit sending the data to an application

```
!return_to_app
```

Situations where multiple policies apply to a single piece of data can be handled using policy intersection. For example, a data subject might state that their raw location data must be anonymized before it is returned to an application but that whether or not they are inside the specified geofence may be shared with an application; however, contractual requirements might independently impose the restriction that no identifiable data may be shared with third parties. The policy expressing how this data may be used would be expressed as the intersection of these two policies

```
((anon + in_geofence) & anon) . return_to_app
```

Note that this policy authorizes execution of the command `anon`, since it satisfies both component policies; it does not authorize the command `in_geofence`.

Finally, a policy might want to allow the same command to be executed any number of times; this authorization is expressed with the Kleene operator. For example, the policy `ANYF*` is associated with public data: it authorizes any sequence of commands to be applied to that data.

Additionally, we define the notation $\mathbb{0}$ to denote the policy that authorizes no programs (that is, $\mathcal{L}(\mathbb{0}) = \emptyset$), and we define the notation $\mathbb{1}$ as syntactic sugar for $\mathbb{0}^*$, which is a policy that authorizes only the empty program with no commands (i.e., $\mathcal{L}(\mathbb{1}) = \{\epsilon\}$).

Rather than explicitly including invokers in a policy, Ancile associates policies with individual applications. When executing a program on behalf on an application, any data fetched by that program is associated with the policy defined for that data provider-application pair. Data values for which no policy is explicitly specified are implicitly associated with the public policy `ANYF*`.

To meet the privacy challenges that arise in applying use-based privacy to location-based services, the Ancile policy language also includes four advanced features: argument-dependent commands, conditions, aggregate transformations, and collections.

Argument-dependent commands: To meet Privacy Challenge 1, we need to allow a policy to specify not only the command but also to specify restrictions on arguments to that command. For

example, given a command `fuzz_location` that takes a mean and a standard deviation—and returns a fuzzed location defined by adding a random value (drawn from the normal distribution with the specified mean and standard deviation), the policy might want to authorize only calls to the command `fuzz_location` where the mean is zero and the standard deviation is at least 10. Accordingly, Ancile policies can place constraints on parameter values. So, for example, a BookNearMe user might associate the following policy with their location data:

```
fuzz_location(mean=0,std>=10) . return_to_app
```

Conditions: In some cases, authorizations depend on context. This context might be value dependent, for example, a RoamingOfficeHours user might want to share their location only if they are currently on campus. This context might even depend on other data values. For example, that user might want to share their location only if they are currently scheduled to hold office hours. Or this context might dependent on public system state, for example, that user might want to share their location only if the current time is during business hours. To express such preferences, Ancile policies may include conditions. A condition command executes a specified predicate (e.g., `in_geofence_cond`). We also introduce auxiliary commands `_test_True` and `_test_False`. So, for example, a user could enforce that the RoamingOfficeHours app only releases their location while they are on campus by defining a policy

```
in_geofence_cond(geofence=GF) .
 ( _test_True . return_to_app +
   _test_False . 0)
```

Observe that conditions are uses: when the predicate is evaluated, the policy on the data value x is modified by taking the derivative with respect to the commands

```
in_geofence_cond(geofence=GF) . _test_True
```

or

```
in_geofence_cond(geofence=GF) . _test_False
```

depending on whether the predicate `in_geofence_cond` evaluates to True or False. Like the use `return_to_app`, conditions have a side effect: they exfiltrate a value from Ancile and send it to the application. However, instead of sending the input value, conditions send the Boolean value the predicate evaluates to.

Aggregate Transformations: To support functions that take multiple arguments, we introduce aggregate transformations, which combine multiple data values $x_1, \ldots, x_n$ into a single data value $f(x_1, \ldots, x_n)$. The policy associated with the new value is obtained by intersecting the derivative policies for input value with respect to f. That is, if $x_1, \ldots, x_n$ have policies $P_1, \ldots, P_n$ respectively, then the aggregate value $f(x_1, \ldots, x_n)$ is associated with the policy $D(P_1, f) \& \ldots \& D(P_n, f)$. For example, suppose that Alice and Bob form a two-member study group, and Alice associates policy

```
in_geofence . evaluate_quorum . return_to_app
```

to her location data, and Bob associates the policy

```
in_geofence . evaluate_quorum .ANYF*.return_to_app
```

to his location data. The application first invokes the command `in_geofence` on each location, yielding Boolean values with respective derivative policies and then performs the aggregate transformation `evaluate_quorum` on the resulting values, yielding a Boolean value with the policy:

```
return_to_app & (ANYF* . return_to_app)
```

As this policy allows calls to `return_to_app`, the resulting value will then be returned to the GroupStudy application, which will notify both Alice and Bob if a quorum is present.

Collections: We define a *Collection* class that stores multiple data values with individual policies. A collection is a policy-protected data structure, with the policy defined as the intersection of the policies associated with the data values in the collection, similar to `aggregate`. But in contrast to a aggregate values, Ancile also tracks the individual policies of each data value in a *Collection*. This allows Ancile to support operations that remove elements from a collection (and synthesize a precise policy for the smaller collection) and to support operations that extract a single element from the collection (and admit policies that maintain the invariant that if value is added to a collection and then removed from the collection, the final policy associated with that value is the same as the initial policy associated with that value).

To support this functionality, we introduce two new transformations: `add_to_collection` takes a collection and one or more additional values as arguments and returns a new collection containing all the values, and `remove_from_collection` takes a collection and an index and removes the value at that index, returning it as the final result.

Other commands also take collections as inputs. Ancile supports many standard transformations on collections, such as `map`, `reduce`, and `filter`. The `map` and `reduce` functions are treated in the same way as aggregation functions—i.e., the command is authorized on the collection only if it is authorized on all values in the collection, but `filter` is handled differently. To authorize the `filter` command, a policy must specify the intended behavior for two cases: `filter_keep` and `filter_remove`. For example, a datapoint might be associated with the following policy:

```
add_to_collection .
  (add_to_collection + filter_keep)* .
   ((average + min) . return_to_app +
      filter_remove . ANYF*)
```

This policy allows the datapoint to be added to a collection, and it allows other datapoints to be added to the collection afterward. For collections that contain this datapoint, the functions `average` and `min` may be computed on that collection (and the resulting outputs returned), but no other functions (other than filter functions and adding other datapoints) may be applied to the collection. After this datapoint is removed from the collection, this policy imposes no further restrictions on how the collection may be used.

4 IMPLEMENTATION

Ancile is designed as a run-time monitor positioned between ubiquitous computing applications and passively-generated data. Applications submit requests to Ancile; each request contains a program to be executed in Ancile's trusted environment along with credentials

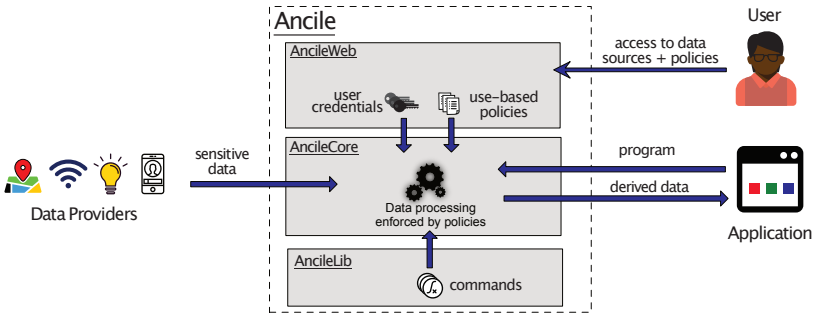

Figure 3: Ancile System.

to authenticate the application to Ancile. Ancile fetches data from a data provider, executes the program, and sends output data to the application if and only if all commands are authorized.

4.1 Trust Assumptions

We assume user locations are collected and stored externally by a third-party data provider, such as an indoor location tracking service. Users have access to their data and may also authorize principals such as Ancile to access their data (e.g., using OAuth2).

We assume that adversaries are applications that consume data to provide some service (e.g., to book rooms near a user). These applications might attempt to perform unauthorized commands on data. We assume that applications do not exploit vulnerabilities in system code, and do not attempt denial of service attacks.

Ancile is a trusted principal. We assume that users trust Ancile with full access to their data. In particular, users trust Ancile to only invoke commands on behalf of an application if those commands are authorized, and to only send data to an application if that release is authorized. We envision two possible ways in which Ancile might be deployed: it might be operated as a trusted third party, or companies might deploy an internal version of Ancile to prevent accidental misuse of data.

4.2 Ancile Overview

Ancile comprises three modules. *AncileWeb* implements a web interface for specifyings policies and integrating data providers. *AncileLib* provides a library of privileged commands that applications use to implement programs. *AncileCore* executes programs on behalf of applications while enforcing policy compliance. The mechanisms of AncileCore ensure that programs cannot violate a user's policy. An overview of the system in shown in Figure 3.

We implemented Ancile in Python 3.7 using the Django web framework [16] to process application requests, control access to data sources, and perform user management. Ancile utilizes the PostgreSQL v11 database [48] to store account credentials, Redis v4.0 [52] to enable in-memory caching of user data and requests, the Gunicorn WSGI server [20], and NGINX reverse web proxy.

4.3 AncileWeb

A user first creates an account on Ancile, via the AncileWeb interface. During *data provider registration*, the user links their account to an external data provider (e.g., a location server) by authenticating to those services. AncileWeb stores delegated authorization credentials (e.g., OAuth2 tokens) on behalf of the user.

As Ancile is designed to support use-based privacy for passively-generated data, it needs a mechanism for policies to get associated with that data. AncileWeb provides an interface for users to specify policies that will apply to all data imported from a data provider; to distinguish between different applications, the user is allowed to specify one policy per data provider-application pair. Each policy specifies which sequences of commands that application is authorized to invoke on any data imported from that data provider. Note, that this implies that all values about one user fetched from one data provider by one application will have the same policy. If a user wants to express different authorizations for different values, they can do so by putting a condition at the beginning of their policy. For example, to distinguish between historical and current traces one can define the following policy:

```
is_current_cond.(_test_True.ANYF* + _test_False.∅)
```

Similarly, if a user wants to distinguish between a single location value and a location trace, they can to do by putting a transformation at the beginning. For example:

```
create_trace . ∅ + !create_trace . return_to_app
```

Currently, policies are defined manually using the syntax described in Section 3. However, in the future, we envision users choosing from a small number of predefined policies created by a *policy administrator*. Ancile policy administrators are also authorized to add policies for any Ancile user or for groups of Ancile users. If no policy is defined for a data provider-application pair, Ancile prohibits all uses of that data by that application.

Since Ancile policies authorize data use for specific applications, Ancile must be able to authenticate applications. Applications register with Ancile through AncileWeb. Once approved, they receive a JSON Web Token (JWT) [29] that authenticates them to Ancile.

4.4 AncileLib

Policies are specified as regular expressions over commands; *AncileLib* provides implementations of those commands organized as Python modules; there is a module for each data provider registered with Ancile. We chose the Python language because it is one of the most common programming languages [50], thereby allowing us to support a wide range of applications.

AncileLib commands may be called by application programs, and the commands are then executed by Ancile on behalf of the application. Each call to an AncileLib command interrupts program execution and invokes AncileCore, a reference monitor that enforces policy compliance before allowing the command to proceed. We use Python decorators to instrument the program so it performs the necessary checks.

AncileLib commands have four different types. Three types were introduced in Section 3: transformations (both basic transformations and aggregate transformations), conditions (which are uses), and returns (which are also uses). A fourth type of of command, called an *external command*, imports data from a data provider into Ancile. These commands operate as follows:

(1) *Transformation commands* take one or more data values as input and return a single derived value as output. A transformation should only be executed if it is authorized by the policy associated with the input values, so transformations include a reference monitor hook—the decorator `@transform`—that invokes the AncileCore reference monitor to check for policy compliance before the command is executed. In a Python implementation of a transformation, `return` sends the data value to the AncileCore monitor that synthesizes a policy for that value. See Figure 4a for an example.
(2) *Condition commands* take a data value as input and evaluate some predicate. Conditions are a type of use, which means that they should only be performed if authorized. Moreover, calling a condition might modify the policy associated with the input value, so a call to a condition includes the decorator `@condition_use`, which invokes AncileCore. AncileCore also updates the policy associated with the input data value. In a Python implementation of a condition, `return` invokes the AncileCore monitor, which exfiltrates the Boolean value (the output of the predicate) to the program. See Figure 4b for an example.
(3) *Return commands* are uses with a side effect: they send the input value to the application. A return should be executed only if authorized, so returns include a reference monitor hook—the `@return_use` decorator—that invokes the reference monitor. AncileCore also updates the policy associated with the input data value. Note that in a Python implementation of a return, `return` sends the value to the AncileCore monitor, which exfiltrates the value to the application. See Figure 4c for an example.
(4) *External commands* receive access tokens from AncileWeb and request data from a data provider. In theory, use-based privacy policies only restrict how data may be used, so Ancile should be allowed to request any data value from any data provider at any time. In practice, however, it is often more convenient to request many data values at the same time (e.g., all data matching a particular query), implicitly aggregating those values together into a single value (e.g., a list). Since user and policy administrators might or might not want to authorize this implicit transformation, external commands include a reference monitor hook—the decorator `@external`—that invokes the AncileCore reference monitor to check for policy compliance before the command is executed. In a Python implementation of an external command, `return` sends the data value to the AncileCore monitor which synthesizes a policy for that data value. See Figure 4d for an example.

Applications use AncileLib commands to write programs that operate on passively-generated data. Applications may implement any program, and these programs may call any command. However, these programs will be executed by Ancile—and successfully complete—only if the sequence of commands called by the application is authorized for that application. For example:

```
data = fetch_data(url=URL, user=user1)
data = fuzz_location(data, mean=0, std=10)
return_to_app(data)
```

This program fetches, fuzzes, and returns the data.

4.5 AncileCore

AncileCore implements a reference monitor that receives and executes programs on behalf of applications while enforcing the restrictions on data use embodied in policies.

Applications primarily communicate with Ancile by making requests for data. When an application requires data from Ancile, it sends a request with the following elements:

(1) Application Token: This secret is used to authenticate the application to Ancile.
(2) Users: The users that the application is requesting data for.
(3) Program: A piece of computation to be executed within Ancile and whose result, if policy compliant, will be returned to the application.

When AncileCore receives a request, it communicates with AncileWeb to authenticate the application. After successful authentication, AncileCore executes that program on behalf of the application while enforcing the associated policies.

Policy enforcement in Ancile is achieved because AncileCore extends programs that operate on data values to be programs that operate on *tagged values* known as *DataPolicyPairs*. A DataPolicyPair contains two restricted fields: `_data` and `_policy`. To prevent programs from directly manipulating data or policies, submitted programs are compiled with *RestrictedPython* [19] before execution. RestrictedPython limits the application's program to predefined Ancile commands and prevents access to internal data structures by transforming the code before compilation and raising errors if a program attempts to use certain built-in features, such as class creation or access protected data fields marked with a leading underscore. In particular, compilation with RestrictedPython guarantees that DataPolicyPairs are opaque to the submitted program and so their internal fields can neither be inspected nor manipulated. Thus, the only way for an application's program to interact with a data value is through AncileLib commands that invoke the reference monitor hooks.

Policy Tagging. There are two ways to create a new DataPolicyPair in Ancile: by importing a raw data value from a data provider and by computing a derived value using a transformation.

Raw values are imported from data providers using external commands. Note that fetching a data value is always authorized. AncileCore determines (1) which user is the subject of that value

```python
@transform
def fuzz_location(data, mean, std):
    import numpy as np

    data['x'] += np.random.normal(mean, std)
    data['y'] += np.random.normal(mean, std)
    return data
```

(a) Transformation Command

```python
@condition_use
def equal_cond(data, key, value):
    return data[key] == value
```

(b) Condition Command

```python
@return_use
def return_to_app(data):
    import json
    return json.dumps(data)
```

(c) Return Command

```python
@external
def fetch_data(url,user):
    import requests

    token = get_token(user)
    header = {"Authorization": "Bearer " + token}
    result = requests.get(url, headers=[header])
    if result.status_code == 200:
        return result.json()
```

(d) External Command

Figure 4: Example commands from AncileLib.

(specified by the request issued by the application), (2) which application is requesting the data (determined from the application token in the request issued by the application), and (3) which data provider acts as the source of the data (determined from the external command). AncileCore then retrieves the corresponding policy from AncileWeb. To execute an external command, AncileCore requests the data value from the external data provider and creates a new DataPolicyPair comprising the value returned by the data provider and the policy returned by AncileWeb.

In practice, it is convenient to allow external commands to fetch multiple data values at the same time (e.g., all data matching a particular query). From a theoretical perspective, these external commands are syntactic sugar for a sequence of requests issued to a data provider (each of which returns a single data value) followed by an aggregation transformation, which combines those values into a single data value. When a multi-value external command is called, AncileCore interacts with AncileWeb to determine the set of implied policies $\mathcal{P}_{imp}$ that would be associated with the imported data values. That is, AncileCore determines which users the application is requesting data (specified by the request issued by the application), which application is requesting the data (determined by the application token in the request issued by the application), and which data provider acts as the source of the data (determined by the external command). AncileCore then invokes its policy enforcement method for the transformations (discussed below) to determine whether the implicit aggregation transformation is authorized. If it is, AncileCore issues a fetch request to the data provider and creates a new DataPolicyPair whose value is the data returned by the data provider and whose policy is the intersection of the derivative policies computed by taking each policy in the set $\mathcal{P}_{imp}$ and computing the derivative with respect to the implicit aggregation transformation.

Derived values are generated from input DataPolicyPairs when transformation commands are called. If authorized, Ancile executes the command, computes the derivative policy of each input, and then creates a new DataPolicyPair comprised of the data value returned by the command and the intersection of the derivative policies for each input (or simply the derivative policy of the one input, if the transformation command has only one input).

In addition to creating new DataPolicyPairs, AncileCore must also modify the policy in an existing DataPolicyPair when a use command is called. For return commands, AncileCore simply replaces the policy in the DataPolicyPair with the derivative of the original policy with respect to the return command. For conditions, it evaluates the specified predicate. It then replaces the policy in the DataPolicyPair with the derivate policy with respect to the condition followed by `_test_True` if the predicated evaluated to true, or `_test_False` otherwise.

Policy Enforcement. Every time the program attempts to call a command, AncileCore checks whether that command is authorized. If a program attempts to call an unauthorized command, AncileCore immediately stops program execution and returns an error message to the application. The checks are performed using syntactic derivatives, as described formally in Appendix A.

5 EVALUATION

To demonstrate functionality (and to demonstrate that Ancile successfully addresses the identified privacy challenges), we implemented the four location-based services described in Section 2.

```
dpp = fetch_last_location(user='user1')
dpp2 = fuzz_location(data=dpp,
                     std=10,
                     mean=0)

return_to_app(data=dpp2)
```

(a) BookNearMe

```
cal_dp = get_calendar_events(user='user1')
loc_dp = get_last_location(user='user1')

if in_geofence_cond(data=loc_dp,
                    geofence=GF):
 if event_occurring_cond(data=cal_dp,
                event_name='Office Hours',
                dependent=loc_dp):

  return_to_app(data=loc_dp)
```

(b) RoamingOfficeHours

```
dp_1 = get_last_location(user='user1')
dp_2 = compute_geofence(data=dp_1,
            lat=0, lon=0, radius=10)
dp_3 = get_last_location(user='user2')
dp_4 = compute_geofence(data=dp_3,lat=0,
                    lon=0, radius=10)

dp_aggr = evaluate_quorum(
              data=[dp_2, dp_4],
              threshold_percent=100)

return_to_app(data=dp_aggr)
```

(c) GroupStudy

```
collection = fetch_location_history(
                    user='user1',
                    fr=DATE_FROM, to=DATE_TO)

filter_train = lambda x:
                   x['timestamp'] <= DATE_TEST
train_data = filter(collection, filter_train)
model = train(data=train_data, epochs=10)

filter_test = lambda x:
                    x['timestamp'] > DATE_TEST
test_data = filter(collection, filter_test)

preds = serve(model=model, data=test_data)
return_to_app(data=preds)
```

(d) LocationPredictor

Figure 5: Ancile programs for example location-based services.

The BookNearMe, RoamingOfficeHours, and GroupStudy are built as Slackbot applications (Section 5.1), while LocationPredictor is realized in terms of several different machine learning pipelines (Section 5.2). We also performed a series of benchmarks to evaluate the overhead incurred by Ancile (Section 5.3).

To provide these sample applications with location data, we developed two standalone location servers: one indoor and one outdoor. The indoor location tracking uses a campus-wide deployment of the Aruba WiFi system with enabled positioning service [57] that our server queries every 30 seconds; the outdoor location server fetches data through a companion Android application using Android's location services [4]. Both servers expose OAuth2 protected endpoints that release location data. Additionally, we tested Ancile with third-party data providers for non-location data, including Google and Outlook Calendars.

5.1 Location-Aware Slackbot applications

In our setup, Slackbot applications communicate with the user through the Slack API and can only access users' data through Ancile. The privacy policies shown below are constructed by the policy administrator. These policies do not block the applications' main functionality, but they enhance privacy by restricting unnecessary uses.

BookNearMe: Our location server data provider returns the current indoor position of the user. Our goal is to prevent the application from learning an exact location, but provide a location sufficient to decide on nearby meeting rooms. The privacy policy for this application is as follows:

```
fuzz_location(std>=10, mean=0) . return_to_app
```

This policy authorizes execution of the `fuzz_location` command to add Gaussian noise to the indoor position; the reactive nature of our policies enforces that data cannot be returned to the application until it has been fuzzed by this command. Note that this policy

only authorizes the `fuzz_location` command when called with a standard deviation greater than or equal to ten and a mean of zero.

An Ancile program that would comply with this policy is shown in Figure 5a. Calling external command `fetch_last_location` returns a new DataPolicyPair, `dpp`, containing the most recent location value from the indoor location service and the policy shown above. Any application wishing to get location data must invoke the `fuzz_location` command with appropriate parameters. This command transforms the location data and returns the fuzzed location in a new DataPolicyPair `dpp2` associated with the derived policy `return_to_app`. The program is then authorized to invoke the return command `return_to_app` on `dpp2` to send the fuzzed location back to the application, which can use this data to book a nearby meeting room on behalf of the user.

RoamingOfficeHours: This application requires access to both calendar and location data, and the policy protecting location data is *dependent* on the calendar data.

This application uses the outdoor location server to fetch data using the command `get_last_location`. The `in_geofence` command determines if the user is in the specified geofence. In this scenario, we want to release the exact location only when the user is inside the specified geofence and office hours are occurring. Thus, we define the following policy on location:

```
in_geofence_cond(geofence=GF) .
  _test_True
  event_occurring_cond(event_name='Office Hours',
                       calendar='user1') .
  _test_True . return_to_app
```

As this application uses calendar data in addition to location data, users or policy administrators will also need to define a policy for how calendar data may be used. If users only care about privacy for location data, they could associate the calendar data provider with the public policy `ANYF`*. Alternatively, if they only want their calendar to be used to check for office hours, they could associate the calendar data provider with the restrictive policy:

```
event_occurring_cond(event_name='Office Hours')*
```

There is no return command in this policy, so calendar data is never sent directly to the application. Instead, calendar data may only be used to determine whether office hours are currently scheduled.

A program that implements the core functionality of the RoamingOfficeHours application and that complies with these policies is given in Figure 5b. The program retrieves both data values and evaluates the conditionals in sequence. The program returns location data only when both predicates hold.

GroupStudy: This application aggregates data from a predefined group of users. We use the `compute_geofence` transformation that takes the raw location and outputs one of the specified geofences. Aggregate transformation `evaluate_quorum` takes a list of datapoints, a threshold, and a list of users in the group and outputs whether the specified users are co-located. The reference monitor checks that data provided to `evaluate_quorum` method belong to the users mentioned in the policy.

```
compute_geofence(lat=0, lon=0, radius=10) .
   evaluate_quorum(threshold_percent=100,
                   users=['user1', 'user2']) .
   return_to_app
```

See Figure 5c for a policy-compliant implementation.

Although these applications are simple, they show how Ancile could be used to support development of a broad range of location-based applications, given a more complete library of commands for common data providers.

5.2 Machine Learning Pipelines

We now consider an application that uses indoor location data to train and use a location-prediction model. We want to control how location data are used individually, how aggregate location traces are used, and how derived machine learning models are used. Ancile collections facilitate implementation of a privacy-enhancing version of the LocationPredictor application.

We might consider four possible approaches to developing the LocationPredictor application with varying levels of privacy protection for the location data used to train the model:

(1) *Release training data to the application.* This approach does not impose any restrictions on data use. A user comfortable with releasing location data to the application might use the `ANYF`* policy, which treats all data as public.

(2) *Train the model inside Ancile and release the model to the application.* Even if a user is unwilling to release raw location data to an application, that user might be willing to allow the application to receive a machine learning model trained on location data. Such a user might use the following policy,

```
add_to_collection . filter_keep* .
  train . return_to_app
```

which only permits use of the model not the training data.

(3) *Train the model inside Ancile and release predictions to the application.* Existing attacks can perform membership inference on training data and even extract data [6], so some users might not want to release a model (trained on their location data) to an application. However, those users might allow Ancile to train a model on location data and to use current data to predict future locations. This policy would be expressed as:

```
add_to_collection . filter_keep* .
   (train.serve + serve) . return_to_app
```

An aggregation function `serve` takes the trained model and the data and returns the predicted future location. Note that this policy does not allow direct use of the model or training data.

(4) *Train the model using a differentially-private mechanism.* There exist model inference attacks that learn a model given only black-box access to the model, so some users might not be comfortable releasing predictions based on a standard model trained on their location data. Such users might require that the model be trained using a differentially-private mechanism that ensures privacy of individual data values [1, 40]. The following policy enforces this case:

```
add_to_collection . filter_keep* .
  train_dp(eps<10) . return_to_app
```

This policy requires use of a specific differentially-private training algorithm.

We implemented variants of the LocationPredictor application that satisfy each of the four proposed policies described above. We used one of the authors' location trace containing three months of location data collected by our indoor location server (a campus-wide deployment of the Aruba WiFi system with enabled positioning services). The location trace contains 29K datapoints that represent 118 distinct locations. For the LocationPredictor application we built a model that predicts the next location given the 20 most recent locations. The model shares structure and hyperparameters with the next-word prediction example from the PyTorch repository [51]. We implemented the normal training of the model as an AncileLib command `train` and used differentially-private version of the SGD algorithm, DP-SGD [40], for `train_dp`. The normal training in cases (1), (2), and (3) achieves 85% accuracy on test data. Training a model in case (4) achieves 75% accuracy and represents a ($\epsilon = 2.11$, $\delta = 10^{-6}$)-DP mechanism (single digit ϵ values provide acceptable guarantees [1]). The program that satisfies the policy (3) is given in Figure 5d.

Encryption. To support applications that want to use the same model during multiple requests, Ancile allows encrypted copies of the model to be returned to the application. Encrypted copies can be sent back to Ancile with future requests. This enables a model trained during one request to be used on new location values in a subsequent request.

Third-party libraries. As in the example above, a policy might require complex transformations to be performed on data, such as computing certain statistics using data science tools, that are expensive to re-implement as AncileLib commands. Ancile can treat methods from *trusted* third-party libraries (e.g. NumPy [45] or PyTorch) as transformation commands, hence library methods can accept DataPolicyPairs as arguments and also transform policies.

5.3 System Performance

Interposing Ancile between applications and data sources adds a layer of indirection that impacts when applications receive data. The latency of requests to Ancile can vary greatly, depending on the executed program and the latency of data providers. In many cases, much of the computation done by Ancile—such as calculating geofences or training machine learning models—would otherwise fall to the application; the complexity of these computations cannot be controlled by Ancile. Similarly, the latency from data providers is unpredictable, often exceeding several seconds, and equally unavoidable. Thus, we focus on measuring the time needed to retrieve policies and data and execute policy checks.

We benchmarked the policy evaluation time for our example applications. The time to evaluate a single policy ranges between 1 to 15 microseconds based on the complexity of the policy, introducing negligible overhead. The other source of overhead comes in fetching the corresponding credentials, compiling programs, and parsing policies, which on average ranges between 30 to 90 milliseconds depending on the number of users and length of submitted program. However, we cache user credentials, compiled programs, and parsed policies, which reduces overhead to between 3 to 9 milliseconds for subsequent requests. Compared to the latency of retrieving data and executing commands, policy enforcement does not add a significant delay in typical applications.

To test the scalability of our system, we performed concurrent load testing of Ancile using the wrk2 benchmarking tool [66]. We tested on a virtual machine running Ubuntu 18.04 with 8 Gb of RAM and 4 Intel(R) Xeon(R) CPU E5-2620 2.1 GHz processors. To eliminate impact of data source latency, we use static sample data with simple policy: `ANYF` and a simple program that fetches the test data and returns. Without caching, the system can handle up to 200 requests per second. However, with caching enabled, the system can handle 700 requests per second, with an average response latency of 428 milliseconds. Given the settings where we intend to deploy Ancile, we believe the performance of our prototype system is already sufficient to support many applications that poll data at regular intervals.

6 RELATED WORK

Ancile extends privacy research that aims to control application access to users' sensitive data [12, 26, 27, 34, 35, 47, 54]. We compare our framework with solutions that analyze or control data usage.

Policy-based systems: The recently proposed Almond system [11] allows users to express policies using natural language which is later converted into programs that control access to data. Almond focuses on translating policies, whereas Ancile adds policies to application programs directly and allows control over data uses. The privacy-enforcing language Jeeves [68] enables enforcement of policies that access particular fields in an application's program. Instead, Ancile supports reactive policies that change as commands are executed, using policies that are attached directly to data.

The Pilot policy language [46] has a similar integration of a policy language, but uses static analysis of submitted code, whereas Ancile policy enforcement is interleaved with the execution of an application's program and can change based not only commands but also on data. While the Houdini project [27] supports context-aware data sharing, it does not support reactive privacy policies. Decentralized policy enforcement [30] can be further applied to Ancile and increase range of supported applications. The Open Algorithms project [22] proposes a system similar to AncileLib that contains trusted implementations of data processing, but lacks a formal policy language to enforce control over data.

Inspection-based systems: PrivacyStreams [38] integrates into the development flow of Android applications. However, it lacks a policy enforcement component and can only report performed data usage. The TaintDroid [18] and FlowDroid [5] projects can infer an application's usage of sensitive data without access to source code, but cannot enforce policy restrictions. Similarly, data inspection projects [23, 37, 49, 61] only track usage but do not support policy control. On the other hand, ProtectMyPrivacy [2] allows one to implement access protection on data sources, but cannot act dynamically and does not impose usage control.

Personal private spaces: Systems such as Databox [41] and openPDS [15] implement private storage for sensitive data or a Personal Data Space [36]. Databox requires applications to run locally, and openPDS only releases an "answer" to data queries. Instead, Ancile returns transformed data to external applications outside of the

trusted environment, allowing arbitrary programs and guaranteeing data release according to defined policies.

Data Flow Control Systems: Projects focusing on ensuring information flow security [43, 53] do not focus on privacy and reactive policies. Usage Control (UCON) [47] and Privacy Proxy [35] extend a traditional access-based approach but lack reactive policy changes. Thoth [17] and Grok [56] operate on the data provider side and focus on high-performance computing, but do not allow for the integration of policies inside program execution. Ancile, in contrast, focuses on deployment within enterprises dealing with users' sensitive data and assumes no changes to data provider work flow. Software Guard Extensions (SGX) [3, 14] provide additional guarantees for safe execution of programs in untrusted environments. In our current work, we don't consider SGX-based policy enforcement [8, 28, 39, 55] and assume Ancile commands have been inspected and are run in a trusted environment.

Privacy in ubiquitous systems: Sensitive data generated by ubiquitous sensors have been shown to reveal details such as behavioral patterns [21, 25, 64] and physical presence [62, 69, 70] and can lead to stalking or disparate treatment [21, 65] and have been extensively studied [59, 60]. In our experiments, we focus on location data because it is one of the commonly-used sensors for privacy research and it has been extensively studied over last two decades [7, 33, 44]. We use common techniques for data filtering and controlled data release to experiment with potential applications that preserve users' privacy. More advanced techniques of location obfuscation [58, 63] are not considered in this paper, but since Ancile supports adding wide range of commands, it is easy to extend Ancile in this manner.

7 CONCLUSION AND FUTURE WORK

We explored the problem of applying use-based privacy to passively-generated data. Using location-based services as an example, we identified privacy challenges that arise in ubiquitous computing applications, extended the existing Avenance language to address these challenges, and implemented a framework for enforcing use-based privacy in ubiquitous computing applications.

This work constitutes the first evidence that use-based privacy can be leveraged to enhance privacy in ubiquitous computing applications, but it leaves several open questions. First, we hypothesize that the privacy-challenges that arise in location-based services are representative of the challenges that arise in ubiquitous computing applications more broadly. However, this hypothesis is untested to date. The extent to which Ancile solves the problem of applying use-based privacy to the full range of ubiquitous computing applications is left as future work. Second, we believe it would be possible to implement a full data-analytics toolkit in AncileLib that would support a broad range of general-purpose applications that depend on data from many different data providers. However, the current implementation is more tailored to the example applications considered in this work. Future work is needed to confirm that Ancile can support extensible application development. Third, Ancile separates policy from code, relieving application developers of sole responsibility for ensuring that data are only used in compliance with all relevant policies. However, adoption will depend on the ease with which developers can implement new programs that run on top of Ancile. Further evaluation is needed to establish whether Ancile allows non-experts to easily implement privacy-enhancing ubiquitous computing applications.

ACKNOWLEDGEMENTS

The authors would like to thank Arnaud Sahuguet, Ed Kiefer, Mohamad Safadieh, Michael Sobolev, Matthew Griffith, and Corin Rose. This work was supported in part by NSF Grants 1642120 and 1700832. Schneider is also supported in part by AFOSR grant F9550-16-0250.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 308–318.

[2] Yuvraj Agarwal and Malcolm Hall. 2013. ProtectMyPrivacy: detecting and mitigating privacy leaks on iOS devices using crowdsourcing. In *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*. ACM, 97–110.

[3] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13. ACM New York, NY, USA.

[4] Android 2019. Documentation for app developers. https://developer.android.com/docs.

[5] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. 2014. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. *Acm Sigplan Notices* 49, 6 (2014), 259–269.

[6] Giuseppe Ateniese, Giovanni Felici, Luigi V Mancini, Angelo Spognardi, Antonio Villani, and Domenico Vitali. 2013. Hacking smart machines with smarter ones: How to extract meaningful data from machine learning classifiers. *arXiv preprint arXiv:1306.4447* (2013).

[7] Alastair R Beresford and Frank Stajano. 2003. Location privacy in pervasive computing. *IEEE Pervasive computing* 1 (2003), 46–55.

[8] Eleanor Birrell, Anders Gjerdrum, Robbert van Renesse, Håvard Johansen, Dag Johansen, and Fred B Schneider. 2018. SGX Enforcement of Use-Based Privacy. In *Proceedings of the 2018 Workshop on Privacy in the Electronic Society*. ACM, 155–167.

[9] Eleanor Birrell and Fred B Schneider. 2017. *A Reactive Approach for Use-Based Privacy*. Technical Report.

[10] Janusz A Brzozowski. 1964. Derivatives of regular expressions. In *Journal of the ACM*. Citeseer.

[11] Giovanni Campagna, Silei Xu, Rakesh Ramesh, Michael Fischer, and Monica S Lam. 2018. Controlling Fine-Grain Sharing in Natural Language with a Virtual Assistant. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 3 (2018), 95.

[12] Fred H Cate. 2002. Principles for protecting privacy. *Cato J.* 22 (2002), 33.

[13] Fred H Cate, Peter Cullen, and Viktor Mayer-Schonberger. 2013. Data protection principles for the 21st century. (2013).

[14] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* 2016, 086 (2016), 1–118.

[15] Yves-Alexandre De Montjoye, Erez Shmueli, Samuel S Wang, and Alex Sandy Pentland. 2014. openpds: Protecting the privacy of metadata through safeanswers. *PloS one* 9, 7 (2014), e98790.

[16] Django 2019. Django: The Web framework. https://www.djangoproject.com/.

[17] Eslam Elnikety, Aastha Mehta, Anjo Vahldiek-Oberwagner, Deepak Garg, and Peter Druschel. 2016. Thoth: Comprehensive Policy Compliance in Data Retrieval Systems.. In *USENIX Security Symposium*. 637–654.

[18] William Enck, Peter Gilbert, Seungyeop Han, Vasant Tendulkar, Byung-Gon Chun, Landon P Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N Sheth. 2014. TaintDroid: an information-flow tracking system for realtime privacy monitoring on smartphones. *ACM Transactions on Computer Systems (TOCS)* 32, 2 (2014), 5.

[19] Zope Foundation. 2019. RestrictedPython. https://github.com/zopefoundation/RestrictedPython.

[20] Gunicorn 2019. Documentation for Gunicorn. https://gunicorn.org.

[21] Peter Händel, Jens Ohlsson, Martin Ohlsson, Isaac Skog, and Elin Nygren. 2013. Smartphone-based measurement systems for road vehicle traffic monitoring and usage-based insurance. *IEEE systems journal* 8, 4 (2013), 1238–1248.

[22] Thomas Hardjono and Alex Pentland. 2018. Open algorithms for identity federation. In *Future of Information and Communication Conference*. Springer, 24–42.

[23] Alexander Hicks, Vasilios Mavroudis, Mustafa Al-Bassam, Sarah Meiklejohn, and Steven J. Murdoch. 2018. VAMS: Verifiable Auditing of Access to Confidential

Data. *CoRR* abs/1805.04772 (2018). arXiv:1805.04772 http://arxiv.org/abs/1805.04772
[24] Jeffrey Hightower, Sunny Consolvo, Anthony LaMarca, Ian Smith, and Jeff Hughes. 2005. Learning and recognizing the places we go. In *International Conference on Ubiquitous Computing*. Springer, 159–176.
[25] Peter Holley. 2019. Wearable technology started by tracking steps. Soon, it may allow your boss to track your performance. https://wapo.st/2NllTfh.
[26] Jason I Hong and James A Landay. 2004. An architecture for privacy-sensitive ubiquitous computing. In *Proceedings of the 2nd international conference on Mobile systems, applications, and services*. ACM, 177–189.
[27] Richard Hull, Bharat Kumar, Daniel Lieuwen, Peter F Patel-Schneider, Arnaud Sahuguet, Sriram Varadarajan, and Avinash Vyas. 2004. Enabling context-aware and privacy-conscious user data sharing. In *IEEE International Conference on Mobile Data Management, 2004. Proceedings. 2004.* IEEE, 187–198.
[28] Tyler Hunt, Zhiting Zhu, Yuanzhong Xu, Simon Peter, and Emmett Witchel. 2016. Ryoan: A Distributed Sandbox for Untrusted Computation on Secret Data.. In *OSDI*. 533–549.
[29] JWT 2019. JSON Web Tokens. https://jwt.io.
[30] Florian Kelbert and Alexander Pretschner. 2015. A fully decentralized data usage control enforcement infrastructure. In *International Conference on Applied Cryptography and Network Security*. Springer, 409–430.
[31] Daniel Kondor, Behrooz Hashemian, Yves-Alexandre de Montjoye, and Carlo Ratti. 2018. Towards matching user mobility traces in large-scale datasets. *IEEE Transactions on Big Data* (2018).
[32] Elisavet Kozyri and Fred B Schneider. 2019. *RIF: Reactive information flow labels*. Technical Report.
[33] John Krumm. 2009. A survey of computational location privacy. *Personal and Ubiquitous Computing* 13, 6 (2009), 391–399.
[34] Marc Langheinrich. 2001. Privacy by design–principles of privacy-aware ubiquitous systems. In *International conference on Ubiquitous Computing*. Springer, 273–291.
[35] Marc Langheinrich. 2002. A privacy awareness system for ubiquitous computing environments. In *international conference on Ubiquitous Computing*. Springer, 237–245.
[36] Tuukka Lehtiniemi. 2017. Personal Data Spaces: An Intervention in Surveillance Capitalism? *Surveillance & Society* 15, 5 (2017), 626–639.
[37] Tianshi Li, Yuvraj Agarwal, and Jason I Hong. 2018. Coconut: An IDE plugin for developing privacy-friendly apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 2, 4 (2018), 178.
[38] Yuanchun Li, Fanglin Chen, Toby Jia-Jun Li, Yao Guo, Gang Huang, Matthew Fredrikson, Yuvraj Agarwal, and Jason I Hong. 2017. Privacystreams: Enabling transparency in personal data processing for mobile apps. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 1, 3 (2017), 76.
[39] Joshua Lind, Christian Priebe, Divya Muthukumaran, Dan O'Keeffe, P Aublin, Florian Kelbert, Tobias Reiher, David Goltzsche, David Eyers, Rüdiger Kapitza, et al. 2017. Glamdring: Automatic application partitioning for Intel SGX. USENIX.
[40] H Brendan McMahan and Galen Andrew. 2018. A General Approach to Adding Differential Privacy to Iterative Training Procedures. *arXiv preprint arXiv:1812.06210* (2018).
[41] Richard Mortier, Jianxin Zhao, Jon Crowcroft, Liang Wang, Qi Li, Hamed Haddadi, Yousef Amar, Andy Crabtree, James Colley, Tom Lodge, et al. 2016. Personal data management with the databox: What's inside the box?. In *Proceedings of the 2016 ACM Workshop on Cloud-Assisted Networking*. ACM, 49–54.
[42] Craig Mundie. 2014. Privacy Pragmatism; Focus on Data Use, Not Data Collection. *Foreign Aff.* 93 (2014), 28.
[43] Andrew C Myers and Andrew C Myers. 1999. JFlow: Practical mostly-static information flow control. In *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*. ACM, 228–241.
[44] Ginger Myles, Adrian Friday, and Nigel Davies. 2003. Preserving privacy in environments with location-based applications. *IEEE Pervasive Computing* 1 (2003), 56–64.
[45] NumPy 2019. Scientific computing with Python. https://www.numpy.org/.
[46] Raúl Pardo and Daniel Le Métayer. 2019. Analysis of Privacy Policies to Enhance Informed Consent. *arXiv preprint arXiv:1903.06068* (2019).
[47] Jaehong Park and Ravi Sandhu. 2002. Towards usage control models: beyond traditional access control. In *Proceedings of the seventh ACM symposium on Access control models and technologies*. ACM, 57–64.
[48] PostgreSQL 2019. PostgreSQL documentation. https://www.postgresql.org/docs/.
[49] Evangelos Pournaras, Izabela Moise, and Dirk Helbing. 2015. Privacy-preserving ubiquitous social mining via modular and compositional virtual sensors. In *2015 IEEE 29th International Conference on Advanced Information Networking and Applications*. IEEE, 332–338.
[50] PYPL 2019. PopularitY of Programming Language. http://pypl.github.io/PYPL.html.
[51] PyTorch GitHub 2019. https://github.com/pytorch/. [Online; accessed 14-May-2019].
[52] Redis 2019. Redis documentation. https://redis.io/documentation.
[53] Andrei Sabelfeld and Andrew C Myers. 2003. Language-based information-flow security. *IEEE Journal on selected areas in communications* 21, 1 (2003), 5–19.
[54] Vagner Sacramento, Markus Endler, and Fernando N Nascimento. 2005. A privacy service for context-aware mobile computing. In *First International Conference on Security and Privacy for Emerging Areas in Communications Networks (SECURECOMM'05)*. IEEE, 182–193.
[55] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *Security and Privacy (SP), 2015 IEEE Symposium on*. IEEE, 38–54.
[56] Shayak Sen, Saikat Guha, Anupam Datta, Sriram K Rajamani, Janice Tsai, and Jeannette M Wing. 2014. Bootstrapping privacy compliance in big data systems. In *Security and Privacy (SP), 2014 IEEE Symposium on*. IEEE, 327–342.
[57] Company Data Sheet. 2019. ARUBA 310 SERIES ACCESS POINTS. https://www.arubanetworks.com/assets/ds/DS_AP310Series.pdf.
[58] Reza Shokri, George Theodorakopoulos, and Carmela Troncoso. 2017. Privacy games along location traces: A game-theoretic framework for optimizing location privacy. *ACM Transactions on Privacy and Security (TOPS)* 19, 4 (2017), 11.
[59] Eran Toch, Justin Cranshaw, Paul Hankes-Drielsma, Jay Springfield, Patrick Gage Kelley, Lorrie Cranor, Jason Hong, and Norman Sadeh. 2010. Locaccino: a privacy-centric location sharing application. In *Proceedings of the 12th ACM international conference adjunct papers on Ubiquitous computing-Adjunct*. ACM, 381–382.
[60] Janice Y Tsai, Patrick Kelley, Paul Drielsma, Lorrie Faith Cranor, Jason Hong, and Norman Sadeh. 2009. Who's viewed you?: the impact of feedback in a mobile location-sharing application. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2003–2012.
[61] Haoyu Wang, Jason Hong, and Yao Guo. 2015. Using text mining to infer the purpose of permission use in mobile apps. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 1107–1118.
[62] Hao Wang, Daqing Zhang, Junyi Ma, Yasha Wang, Yuxiang Wang, Dan Wu, Tao Gu, and Bing Xie. 2016. Human respiration detection with commodity wifi devices: do user location and body orientation matter?. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*. ACM, 25–36.
[63] Leye Wang, Dingqi Yang, Xiao Han, Tianben Wang, Daqing Zhang, and Xiaojuan Ma. 2017. Location privacy-preserving task allocation for mobile crowdsensing with differential geo-obfuscation. In *Proceedings of the 26th International Conference on World Wide Web*. International World Wide Web Conferences Steering Committee, 627–636.
[64] Stephen B Wicker. 2012. The loss of location privacy in the cellular age. *Commun. ACM* 55, 8 (2012), 60–68.
[65] Delanie Woodlock. 2017. The Abuse of Technology in Domestic Violence and Stalking. *Violence Against Women* 23, 5 (2017), 584–602. https://doi.org/10.1177/1077801216646277
[66] wrk2 2019. Modern HTTP benchmarking tool. https://github.com/giltene/wrk2.
[67] Jie Xiong and Kyle Jamieson. 2013. ArrayTrack: a fine-grained indoor location system. Usenix.
[68] Jean Yang, Kuat Yessenov, and Armando Solar-Lezama. 2012. A language for automatically enforcing privacy policies. In *ACM SIGPLAN Notices*, Vol. 47. ACM, 85–96.
[69] Daqing Zhang, Hao Wang, and Dan Wu. 2017. Toward centimeter-scale human activity sensing with Wi-Fi signals. *Computer* 50, 1 (2017), 48–57.
[70] Yongpan Zou, Weifeng Liu, Kaishun Wu, and Lionel M Ni. 2017. Wi-fi radar: Recognizing human behavior with commodity wi-fi. *IEEE Communications Magazine* 55, 10 (2017), 105–111.

A FORMALIZING POLICY ENFORCEMENT

To implement policy enforcement in Ancile, we use syntactic implementations of an emptiness check and the derivative operation. The emptiness check $E(P)$ produces a Boolean value indicating whether or not the value can currently be used. Formally, it checks whether the language $\mathcal{L}(P)$ generated by the policy P contains the empty string ϵ. By definition, $\mathcal{L}(\mathbb{0}) = \emptyset$, so $E(\mathbb{0}) = 0$. A policy defined by a single command $P = C$ requires that command to be invoked on the data, so $E(C) = 0$. The policy $P_1.P_2$ accepts the empty string only if both P_1 and P_2 do so, thus $E(P_1.P_2) = E(P_1) \wedge E(P_2)$. Union, intersection, and negation are defined in the natural way. An iterated policy P^* accepts any number of iterations of P, including zero (i.e., the empty string ε), so $E(P^*) = 1$. A summary of the operation E is given in Figure 6.

$$
\begin{aligned}
E(\mathbb{0}) &= 0 \\
E(C) &= 0 \\
E(P_1 \ . \ P_2) &= E(P_1) \ \wedge \ E(P_2) \\
E(P_1 \ + \ P_2) &= E(P_1) \ \vee \ E(P_2) \\
E(P_1 \ \& \ P_2) &= E(P_1) \ \wedge \ E(P_2) \\
E(P^*) &= 1 \\
E(!P) &= \ !E(P)
\end{aligned}
$$

Figure 6: Emptiness check operation $E(P)$

$$
\begin{aligned}
D(\mathbb{0}, C) &= \mathbb{0} \\
D(C, C) &= \mathbb{1} \\
D(C, C') &= \mathbb{0} \ \textbf{(for } C \neq C') \\
D(P_1 \ . \ P_2, C) &= D(P_1, C) \ . \ P_2 \ + \ E(P_1) \ . \ D(P_2, C) \\
D(P_1 \ + \ P_2, C) &= D(P_1, C) \ + \ D(P_2, C) \\
D(P1 \ \& \ P2, C) &= D(P_1, C) \ \& \ D(P_2, C) \\
D(P^*, C) &= D(P, C) \ . \ P^* \\
D(!P, C) &= \ !D(P, C)
\end{aligned}
$$

Figure 7: A summary of the syntactic operation D

We can now formalize how Ancile tracks the policy associated with each data value. Ancile executes programs (i.e., sequences of commands) on behalf of applications. When it executes a use $u(x)$, it updates the policy associated with the input x to be the derivative [10] $D(P_x, u)$, where P_x is the policy associated with x before the use u occurs. The formal definition of the derivative policy $D(P, C)$ is given in Figure 7.

Note that if the derivative policy $D(P, c)$ for a command c evaluates to the policy $\mathbb{0}$, that command will continue to be unauthorized at all points in the future. Hence, as an optimization, Ancile blocks any unauthorized commands and terminates the program that attempted to execute that command.

Example. Consider the policy P_0 = anon.`return_to_app` associated with a data value x, which requires that x must be to de-identified (anon) before it may be sent to the application (`return_to_app`).

When the application submits a program that executes the command anon followed by the command `return_to_app`, Ancile system will compute the following derivative policy P_1 to associate with the derived data value anon(x).

$$
\begin{aligned}
P_1 &= D(P_0, \mathsf{anon}) \\
&= D(\mathsf{anon.return\_to\_app}, \mathsf{anon}) \\
&= D(\mathsf{anon}, \mathsf{anon}).\mathsf{return\_to\_app} \\
&\qquad + E(\mathsf{anon}).D(\mathsf{return\_to\_app}, \mathsf{anon}) \\
&= \mathbb{1}.\mathsf{anon} + \mathbb{0}.D(\mathsf{anon.anon}) \\
&= \mathsf{anon} + \mathbb{0} \\
&= \mathsf{anon}
\end{aligned}
$$

When the program executes the second command `return_to_app`, Ancile will compute the derivative policy P_2 and associate it with the value anon(v):

$$
\begin{aligned}
P_2 &= D(P_1, \mathsf{return\_to\_app}) \\
&= D(\mathsf{return\_to\_app}, \mathsf{return\_to\_app}) \\
&= \mathbb{1}
\end{aligned}
$$

Observe that the command `return_to_app` is authorized because $E(P_2) = 1$.

Note that we are using the equation $\mathbb{1}.P{=}P$ (which holds because the policy $\mathbb{1}$ accepts exactly the empty string) the equation $\mathbb{0}.P{=}\mathbb{0}$ (which holds because the policy $\mathbb{0}$ rejects all strings), and the equation $P + \mathbb{0} = P$ (which holds as + corresponds to union).