Privacy Smells: Detecting Privacy Problems in Cloud Architectures

Immanuel Kunz, Angelika Schneider and Christian Banse
Fraunhofer AISEC
Garching b. München, Germany
{firstname.lastname}@aisec.fraunhofer.de

***Abstract*—Many organizations are still reluctant to move sensitive data to the cloud. Moreover, data protection regulations have established considerable punishments for violations of privacy and security requirements. *Privacy*, however, is a concept that is difficult to measure and to demonstrate. While many privacy design strategies, tactics and patterns have been proposed for privacy-preserving system design, it is difficult to evaluate an existing system with regards to whether these strategies have or have not appropriately been implemented. In this paper we propose indicators for a system's non-compliance with privacy design strategies, called *privacy smells*. To that end we first identify concrete metrics that measure certain aspects of existing privacy design strategies. We then define smells based on these metrics and discuss their limitations and usefulness. We identify these indicators on two levels of a cloud system: the data flow level and the access control level. Using a cloud system built in Microsoft Azure we show how the metrics can be measured technically and discuss the differences to other cloud providers, namely Amazon Web Services and Google Cloud Platform. We argue that while it is difficult to evaluate the privacy-awareness in a cloud system overall, certain privacy aspects in cloud systems can be mapped to useful metrics that can indicate underlying privacy problems. With this approach we aim at enabling cloud users and auditors to detect deep-rooted privacy problems in cloud systems.**

***Index Terms*—Data Flow Analysis, Cloud Privacy, Privacy Metrics, Quantifying Privacy**

I. Introduction

Cloud services are often used as backends for collecting and processing data in commercial service offerings. From a privacy-perspective, both the design of a new cloud system as well as the evaluation of an existing system are complex tasks since privacy is an overarching concept that needs to be embedded in different levels of such a system, for instance regarding data sets, data flows and access control. This challenge, together with security concerns, still makes organizations reluctant to adopt cloud computing. Moreover, data protection regulations, such as the General Data Protection Regulation (GDPR), have established standards for security and privacy that need to be adhered to and make violations of respective controls punishable by considerable fines.

Various approaches for designing privacy-aware systems have been proposed, for example risk-driven privacy requirements engineering [1]. In an existing system, however, it is difficult to evaluate whether a privacy-aware design has appropriately been implemented. On the one hand, existing works have proposed high-level descriptions of privacy strategies and patterns for the design of privacy-friendly systems [2] [3]. On the other hand, specific metrics, e.g. for data similarity [4], have been proposed to measure privacy in datasets. It is, however, difficult to measure privacy in a system's implementation, e.g. in data flows between components and in role assignments. Such a measurement would allow a statement about the extent to which privacy strategies are being respected in a system at runtime. We claim that measurable indicators can be found in systems that do not respect these strategies. We call these indicators *privacy smells*.

The concept of *smells* has been introduced by Fowler and Beck [5] for source code. Generally speaking, a smell is a higher-level indication of an underlying problem in a system. When applied to source code, smells may, for example, indicate problems with the underlying software architecture. We apply this concept to privacy identifying smells in a particular domain, namely cloud systems.

We approach the problem of identifying privacy smells by using existing privacy design strategies to infer what measurable indicators may appear in a system which does not respect these strategies. We then show how a privacy smell can be detected based on a certain metric, similarly to how it has been done for code smells as well (e.g. [6]).

Our contributions include the following:

- Metrics and smells for various privacy aspects in cloud architectures, based on data flows and role assignments, and
- a detailed description of how these metrics translate to technical measurements in Microsoft Azure.

In summary, we show that while privacy is difficult to evaluate in a system overall, useful metrics can be found for certain privacy aspects in cloud architectures, e.g. data minimization and access control. We argue that smells as proposed in this paper are significant for a cloud user's or an auditor's ability to find and correct privacy problems that are related to non-compliance with common design strategies.

The remainder of this paper is structured as follows. Section II briefly explains existing privacy design strategies which we use as the basis for our approach. In Section III we deduct concrete metrics from these strategies, define respective smells, and discuss their usefulness in Section IV. We show in Section V how they can technically be measured in Azure and discuss differences to other cloud providers. Section VI describes related work and Section VII concludes the paper.

978-1-6654-0392-4/20/$31.00 ©2020 IEEE
DOI 10.1109/TrustCom50675.2020.00178

II. Background: Engineering Privacy

In this Section we introduce existing strategies for designing privacy-aware systems, namely Hoepman's Privacy Design Strategies [2] as well as design strategies based on the guidelines developed by Hu and Kent [7] for role-based access control (RBAC) systems.

A. Privacy Design Strategies

Hoepman [2] proposes eight high-level privacy design strategies which we describe in the following. We also refer to Colesky et al. [8] who have enhanced these strategies by proposing more concrete *tactics* for every strategy.

1) Minimise: This strategy aims at minimizing the amount of personal information that is collected, processed and disseminated. Concrete tactics in this context can be to completely refrain from the collection of data from certain individuals and to remove personal data, e.g. as soon as it is not needed anymore.

2) Hide: A further strategy is to hide personal data from plain view, i.e. to ensure confidentiality. A useful tactic to that end can be encryption of data and the implementation of access control systems.

3) Separate: Separating the processing and storage of personal data can be a strategy towards unlinkability of data items. Tactics in this context can be to isolate the processing of different parts of data and to distribute the processing and storage rather than centralizing it.

4) Aggregate: This strategy suggests to process personal data on the highest possible level of abstraction, i.e. with the least necessary detail. Colesky et al. mention summarizing data as a possible tactic, i.e. processing correlations of data items rather than the data itself.

5) Inform: Informing data subjects about the processing of their personal data ensures transparency. This strategy, as well as the following three, refers to processes around the actual data collection and processing.

6) Control: This strategy ensures that data subjects keep control over the collection and processing of their data, e.g. by allowing them to modify their personal data.

7) Enforce: The enforce strategy aims at policy compliance through technical and organizational measures. Again, this strategy may cover access control mechanisms, but can cover many more privacy-preserving mechanisms that can be expressed through policies.

8) Demonstrate: Being able to demonstrate compliance with privacy policies and legal requirements is a further design strategy. Concrete tactics can be to conduct audits of the system, creating logs and reporting the results.

B. RBAC Design Strategies

While Hoepman refers to access control in the *Hide* and *Enforce* strategies, more concrete guidelines for designing access control systems can be found in the literature, e.g. proposed by Hu and Kent [7].

Consider first the abstraction of a role assignment, illustrated in Figure 1, which we will refer to in the privacy smells in the next Section. Figure 1 shows the components of a role assignment as it is defined in a RBAC system. A role assignment is composed of an assignee (e.g. a user group) a number of roles which provide permissions to the assignee, and a number of resources to which the permissions extend, i.e. its *resource scope*. As the figure shows, role assignments can be composed of a single user and a single resource rather than a user group and a resource group.

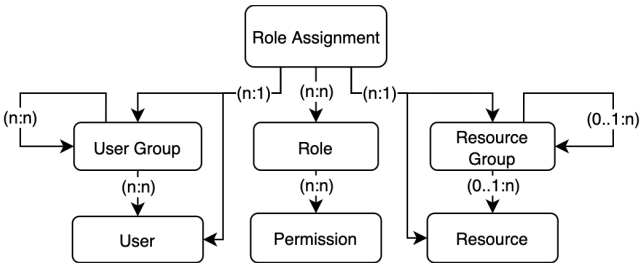

Fig. 1. An overview of the components of a role assignment. Arrows can be read as *is composed of* and are denoted with the composition's multiplicity. For instance, the multiplicity *(n:1)* on the arrow from role assignment to resource is read as *a role assignment is composed of 1 resource; 1 resource can be part of n role assignments*. Note the contrast between user (groups) and resource (groups): while users may be part of many user groups and role assignments, resource groups are strictly hierarchical—a resource is only assigned to at most one resource group.

We base the RBAC smells we propose in Section III on the following two design strategies, proposed by Hu and Kent [7].

1) Least Privilege: This well known strategy aims at giving every user exactly the permissions that the user requires. This can, for instance, mean that custom role assignments are designed for every user.

2) Separation of Duties: The goal of this principle is to prevent users from accessing data items that are not related to their duties. As such, it has a similar goal as the least privilege principle but follows the approach of creating duty-based user groups rather than defining permissions for every single user.

In the following Section we use these strategies to deduce privacy metrics and respective smells for cloud architectures.

III. From Design Strategies to Smells

In the following we propose seven privacy smells and respective metrics that can be used to detect these smells. We follow a common approach to the detection of smells: First, a metric $m(e) = v$ is defined as a function of an element e or a set of elements E that outputs a value $v \epsilon V$. When detecting code smells, for example, the metric may take a method as an input and may output a value for its cyclomatic complexity. Second, a smell threshold $t \epsilon V$ is defined. A smell is then detected when the value exceeds the threshold, i.e. $v > t$. For example, a detection rule may state that a method's cyclomatic complexity of $v > 15$ is regarded as a code smell.

While it is difficult to define precise thresholds that are generally applicable, existing approaches have proposed such thresholds based on an existing benchmark [6]. Note that in this paper we do not propose precise thresholds for the detection of smells since such a benchmark for cloud systems

is not available. Instead we detect a smell either on the higher or on the lower end of a metric. Also, we envision cloud users who apply our metrics to define custom, expected thresholds.

In the description of the smells we use the following terms:

- A *data item* is an abstraction of a collection of data or a single datum generated by an external *data source*. For example, it may be an identifier, but it can also be a file, e.g. a spread sheet created by a user, or a message, e.g. a status update generated by an IoT device.
- *Data sinks* are cloud components which store data items permanently. This can be, for instance, a structured or unstructured database or an object storage.
- *Resource groups* are logical collections of resources in a cloud system. This concept exists in all major cloud platforms, e.g. in Azure, Amazon Web Services (AWS) and Google Cloud Platform (GCP). It can be used to separate resources, for example to form application-centric groups with the goal of separating resources from each other to simplify security and maintainability.
- *User agents* are identities in a cloud system that represent a resource, e.g. a virtual machine, or a human user.

To clarify the scope of our approach we define a cloud system to which our metrics and smells are applicable as follows: The cloud system collects, processes and stores personal data. It is implemented in a public cloud and its access control is managed using a RBAC system, which is the standard access control model in cloud computing. The system's resources are furthermore logically separated into resource groups. Note also that we assume a benign cloud user applying the metrics to find privacy problems in the user's cloud system.

In the following we break down the strategies introduced in Section II to concrete metrics. We do so by first arguing which indicators would appear in a system that does *not* respect a certain strategy and we identify metrics to measure these indicators. This approach results in two categories of smells: data flow smells and role assignment smells. The metrics we propose include functions whose evaluation often depend on specifics of the cloud provider. In Section V we show how these functions can concretely be evaluated in Azure and we discuss some differences to AWS and GCP.

We furthermore illustrate the smells with a running example as follows: Imagine the ride-hailing service *CloudRide* that connects passengers, who want to go from one location to another, with drivers, who offer rides in their personal cars. CloudRide employs a central cloud platform for the coordination of the service and offers a mobile application for users and drivers that collects personal data, such as payment information and the routes a user has driven.

A. Data Hoards

This smell is an indication for data sinks that collect and store data items unnecessarily.

A common design principle for privacy-aware systems is data minimization, i.e. limiting the data collection and processing of personal data to the necessary minimum. A system that respects this strategy would therefore only collect and process personal data that is necessary for its previously defined purpose—whereas a system that does not respect the minimization principle would collect data that is not used afterwards or used inappropriately. While it is difficult to measure if a data item has been used appropriately with respect to the purpose of its collection, it can be measured if a data item has not been accessed at all. To identify such data items we propose to measure the proportion of unaccessed data items within all existing data items in a data sink.

Let $DH(ds)$ denote the data hoard metric that calculates the fraction of data items in a data sink ds that have never been accessed; let furthermore D^{ds} denote the set of data items stored in ds; and let $nacc(d)$ be 1 if data item $d \epsilon D^{ds}$ has never been accessed and 0 otherwise, then:

$$DH(ds) = (1/|D^{ds}|) * \sum_{i=1}^{|D^{ds}|} nacc(d_i) \quad (1)$$

In the running example, imagine CloudRide continuously collects usage data from passengers and drivers, e.g. speed data, with the purpose of analyzing it and improving the cloud system's resource allocation. While the speed data is continued to be collected, the data analysts, however, never access it since it is not deemed useful. Using the proposed metric, CloudRide identifies the respective data sinks that store this unused data.

B. Data Broadcasts

The data broadcasts smell gives an indication for data sinks that disseminate data items unnecessarily.

The *Minimise* strategy does not only refer to data collection and processing, but also to dissemination of data. Two measurable factors can be considered important regarding the dissemination of data items: the number of times an item has been accessed and the number of user agents who have accessed the data item.

From a privacy perspective, the total number a data item has been disseminated does not allow to make an assumption about potential privacy loss since it may still only have been accessed by one authorized agent. A privacy problem may, however, be detected by the number of groups of user agents that have accessed the data item in the past.

Let $DB(ds)$ denote the data broadcast metric that calculates the number of user agent groups which have accessed data items in the data sink ds; and let $acc(g, ds)$ be 1 if at least one user agent from user agent group $g \epsilon G$ has accessed a data item in ds and 0 otherwise, then:

$$DB(ds) = \sum_{i=1}^{|G|} acc(g_i, ds) \quad (2)$$

In our running example, CloudRide allows restricted access to some data sinks to third parties. CloudRide therefore knows how many user groups should have access to data items in a certain data sink. Using the proposed metric, CloudRide detects a data sink that is accessed by more user groups than expected, due to legacy guest accounts.

C. Data Lakes

This smell aims at detecting data sinks that aggregate data items from many different sources, indicating a violation of the *Separate* strategy.

This strategy aims at separating the processing and storage of personal data that originates from different sources. A cloud system that has been designed using this strategy would therefore process and store data from different sources in different backend resource groups. On the contrary, an architecture that does not respect this strategy would exhibit data flows from different resource groups to the same data sink.

Let $DL(ds)$ denote the data lake metric that calculates the number of user agent groups that have stored data items in a data sink $ds \epsilon DS$; and let the function *stored(g, ds)* be 1 if at least one user agent from user agent group $g \epsilon G$ has stored a data item in the data sink ds in the past, then:

$$DL(ds) = \sum_{i=1}^{|G|} stored(g_i, ds) \tag{3}$$

In our running example, CloudRide adds a new car-sharing service where users can use a shared car owned by CloudRide to drive it themselves. It operates its two services on the same cloud platform. CloudRide furthermore collects and stores personal data about driving routes, payment information and more and uses this information in the car-sharing and ride-hailing services. These services are divided into several logically separated resource groups, e.g. business logic, billing and data analysis. CloudRide now uses the proposed metric to identify data sinks that aggregate data from unexpectedly high numbers of resource groups, e.g. due to customer databases that are shared between the services.

Note that cloud providers offer dedicated services called *data lake* where cloud users can store structured and unstructured data to be analyzed later[1][2]. Yet, we think that this exact concept can be problematic from a privacy perspective which is why we call this smell data lakes as well.

D. Exposed Data

This smell indicates a potentially unwanted visibility of data items due to flawed configuration.

The *Hide* strategy aims at ensuring confidentiality of data and mentions encryption as a possible means to achieve this goal. We therefore define a metric for counting unencrypted data items at rest, see (4), as well as unencrypted data items in transit, see (5). Furthermore, many cloud providers offer the functionality to make a data sink public, i.e. accessible without authentication. We therefore propose to measure the number of public data sinks as well, see (6). Since configurations are usually made for a data sink instead of a single data item, we define these metrics not for single data sinks, but for the set of all data sinks DS in a cloud system.

Let the function *nRestEnc(ds)* be 1 if data sink $ds \epsilon DS$ is not configured for encryption at rest and 0 otherwise, then the proportion of data sinks which can contain unencrypted data items $ED_{rest}(DS)$ is defined as:

$$ED_{rest}(DS) = (1/|DS|) * \sum_{i=1}^{|DS|} nRestEnc(ds_i) \tag{4}$$

Let furthermore the function *nTransEnc(ds)* be 1 if data sink $ds \epsilon DS$ accepts unencrypted requests and 0 otherwise, then the proportion of data sinks which can exhibit unencrypted data flows $ED_{trans}(DS)$ is defined as:

$$ED_{trans}(DS) = (1/|DS|) * \sum_{i=1}^{|DS|} nTransEnc(ds_i) \tag{5}$$

Let furthermore the function *public(ds)* be 1 if a data sink $ds \epsilon DS$ accepts un-authenticated, external requests and 0 otherwise, then the proportion of data sinks which are publicly accessible $ED_{pub}(DS)$ is defined as:

$$ED_{pub}(DS) = (1/|DS|) * \sum_{i=1}^{|DS|} public(ds_i) \tag{6}$$

Consider again the running example. As stated earlier, CloudRide collects and stores many personal data items which should all be encrypted at rest as well as in transit. Using the exposed data smell, CloudRide can identify data sinks that are exposed to unauthenticated and unauthorized requests, e.g. because an employee inadvertently created a public data sink.

E. Spaghetti Flows

This smell is related to two design strategies: First, the *Hide* strategy which states that personal data should be hidden from plain view; and second, the *Enforce* strategy which concerns the set up and enforcement of privacy policies. These strategies call for a system design with isolated resource groups that are separated based on their logical function in the system. A cloud system that does not respect these strategies would therefore exhibit data flows that are not processed and stored in a single logical unit, i.e. resource group, of the cloud system but are processed and stored by different units.

Such a flawed assignment of resources to resource groups is not only problematic to manage, but can also undermine a privacy policy that is specified based on resource groups.

Let $SF(d)$ denote the spaghetti flows metric that counts the number of groups that have processed a data item d; and let $acc(r, d)$ be 1 if any resource from resource group $r \epsilon R$ has accessed data item d at least one time, and 0 otherwise, then:

$$SF(d) = \sum_{i=1}^{|R|} acc(r_i, d) \tag{7}$$

In our running example, CloudRide uses different resource groups in different services, e.g. to separate their development, staging and production environments, and to design their access control system accordingly. Using the proposed metric, CloudRide identifies problematic data flows where personal

[1]https://aws.amazon.com/big-data/datalakes-and-analytics

[2]https://azure.microsoft.com/en-us/solutions/data-lake

data items from the production environment have also been used in the development environment.

This smell is related to the commonly used smell for source code, called *spaghetti code*, which identifies badly structured code. We apply this concept to data flows here to identify data items that may violate privacy strategies.

F. Mixed Duties

This smell refers to the separation of concerns between user groups and is based on the guideline of separation of duties (SoD) as described in Section II-B. One common possibility to establish SoD is to create user groups for certain duties. Role assignments can then be designed for these user groups and users who are assigned to these groups automatically inherit their role assignments (see Section II-B). Assuming that the RBAC system is built around such user groups, it can be reasoned that a system that does not respect the SoD principle would exhibit a considerable overlap between user groups in their assigned rights. To identify this overlap we propose to measure the proportion of $(permissions, scope)$-tuples that are shared among all user groups in the RBAC system.

Let $MD(U)$ denote the mixed duties metric that calculates the overlap within the set of all user groups U; let R_i denote the set of role assignments that are associated with the user group i; and let $R_i \cap R_j$ denote the overlap of the permissions of R_i and R_j which are in the same scope, then:

$$MD(U) = (1/|U|) * \sum_{i=1}^{|U|} \sum_{j=1, i \neq j}^{|U|} \frac{|R_i \cap R_j|}{|R_i \cup R_j|} \tag{8}$$

In the running example, CloudRide notices that both data analysts as well as developers have been assigned access rights to a data sink that stores personal user data, although only a part of each group really requires this access for testing purposes. CloudRide therefore moves these permissions to a new, third group so the employees' duties can be separated more granularly.

G. Broad Assignments

This smell refers to the granularity of role assignments regarding their resource scope, e.g. to single resources or resource groups. It is based on the least privilege principle (see Section II-B). Apart from user groups, the resource hierarchy in a cloud system can be used to simplify the assignment of roles to users. This can be done by defining the resource scope of a role assignment on one level of the resource hierarchy, such as a single resource or a resource group. When using a resource group, the role assignment automatically extends to all resources that are part of the resource group, i.e. to all resources downwards in the hierarchy.

Considering an RBAC system that complies with the least privilege principle, it can be reasoned that a relatively small number of role assignments should be associated with a resource scope further up in the resource hierarchy, e.g. for administrators; analogously, a relatively large number of assignments should be defined with a resource scope further down in the resource hierarchy, e.g. for developers of a specific application. Consequently, an RBAC system that exhibits a large relative number of role assignments that are scoped to higher-order resource groups can be considered an indication of an underlying access control and privacy problem.

Let $BA(R)$ denote the broad assignments metric that calculates the average resource scope depth of all role assignments R; and let $scopeDepth(r)$ be the scope depth of a single role assignment $r \epsilon R$, then:

$$BA(R) = (1/|R|) * \sum_{i=1}^{|R|} scopeDepth(r_i) \tag{9}$$

In the running example, CloudRide has scoped the role assignments of different user groups to the resource groups related to their duties since different data analysts, developers etc. are responsible for different resource groups. CloudRide constantly monitors the broad assignments metric and notices a sudden spike in the average scope depth due to flawed role assignments of a user group.

IV. Discussion

In this Section we discuss some limitations of the proposed smells and of our approach in general.

A. Smell Limitations

1) Data hoards: This smell aims at identifying data collections that have been aggregated without a purpose by identifying unused data items. A limitation of this smell is that unused data collections may exist for a good reason, e.g. for redundancy purposes. Yet, we defined this smell's metric on the data sink level, so a cloud user who applies this metric may specifically exclude a backup data sink from the measurement. Also, a data sink may store data items that have just recently been added but not accessed yet. We therefore propose a relative metric to take this into account.

2) Data broadcasts: This smell identifies groups of users and groups of resources that have accessed a data item in the past. This approach is limited by the fact that some resources, like virtual machines, can act on behalf of a user, i.e. there can be an overlap between resource groups and user groups. Also, a user may be part of several user groups that all have the access permission to a data item. This case, however, is a smell in itself and is covered by the mixed duties smell.

3) Data lakes: This smell identifies data collections that have been aggregated from different resource groups. Here, it can be argued that simply storing data items from different sources in one data sink is not a privacy problem until the data items are actually accessed (which is covered by the data broadcasts smell). We would, however, argue that data lakes indicate a flawed system design where data sinks should be split up even if a data broadcast smell cannot yet be detected.

4) Exposed data: This smell identifies data sinks that store visible data items. The respective metrics are limited in cases where data items are made public and left unencrypted on purpose. In this case, however, we would argue that public data

items should be stored together in a public data sink which, again, can easily be excluded from the metric application.

5) Spaghetti flows: This smell identifies unnecessary exposure of data flows to several resource groups. While exposure to several resource groups does not always entail a privacy risk, we would argue that a high measurement value of the respective metric indicates a flawed assignment of resources to resource groups since a data item should belong to one logically separated group of resources. This problem is aggravated if role assignments use broadly assigned resource scopes as described in the broad assignments smell. As such, this smell can be seen as a counterpart to the mixed duties smell, but targeted at resource groups rather than user groups.

6) Mixed duties: This smell identifies overlapping permissions between user groups which can indicate a violation of the separation of duties principle. Such overlapping permissions, however, do not necessarily mean that users have overly broad permissions. Also, there are roles which must have overlapping permissions, for instance a security role which needs read access to all resources in the cloud system.

7) Broad assignments: This smell identifies role assignments with broad resource scopes. A problem with assigning narrow scopes, however, is a more complex maintainability, similarly to defining custom role assignments for every user to implement the least privilege principle.

B. Further limitations and remarks

As noted earlier, we do not provide exact thresholds for the metrics. We think that cloud users who apply the proposed metrics need to be able to define custom thresholds for the detection of smells since thresholds are usually system-specific, e.g. due to different redundancy rules and the number of applications hosted in the same cloud system. We acknowledge that it is not always trivial to define such an expected threshold for the proposed metrics. Yet, we see a second way to use these metrics in the monitoring of the metric values to detect significant changes over time (without defining thresholds).

Furthermore, in the metrics and smells we propose we do not differentiate between personal and non-personal data items since this would require to make assumptions about the availability of meta-information about the data. A cloud user may take this into account by excluding certain data sinks from the application of some metrics.

Our RBAC smells consider two components of a role assignment, namely the separation between user groups, and the scopes depth. We do not propose a smell for the number of permissions a role assignment has since no statement can be made about permissions without also considering their resource scope: a large number of permissions may be scoped to only one resource, while only one permission may be scoped to many resources.

As we show in the next Section, the calculation of the proposed metrics requires that accesses to data items, including the user agent, is logged. Saving these logs can present a privacy problem in itself, e.g. if the access to logs is not properly managed—resulting in mixed duties or a data lake.

V. Measurement in Microsoft Azure

The proposed metrics are not always trivial to measure in concrete cloud systems. Furthermore, their measurement may differ considerably between different cloud providers. In this Section we show how they can practically be measured in Microsoft Azure. We identify four general types of technical measurements that are necessary to compute the functions included in (1)-(9), e.g. analyzing access requests, and describe these in the following Subsections. In the end of the Section we also discuss the differences to AWS and GCP.

We have previously defined and used the abstract terms *data sink* and *data item*; in the following we use *storage accounts* and *blob objects* as their respective equivalences in Azure.

A. Storage Access

To calculate the data hoards, data broadcasts and data lakes metrics, the storage accesses need to be analyzed. This includes the user agent who has made the access and its group associations. In Azure, *Storage Analytics logs* (hereafter called logs) are generated for various kinds of events, including read and write operations on storage accounts. In our implementation we store and analyze the logs in a further Azure service, called *Monitor*[3][4]. Furthermore, we use a Powershell script[5] to convert the CSV-formatted data to JSON and post it to Monitor for analysis. Monitor provides a custom SQL-like query language, called Kusto[6], that can then be used to analyze the logs.

1) Data hoards: For the data hoards metric, it needs to be checked if a data item has been accessed or not, see (1). Using Kusto we simply count the read requests made per blob object and filter for objects which have a count of zero.

2) Data broadcasts: For the data broadcasts metric, see (2), we need to analyze which user agent or group accessed a blob object. To do so we first use a Kusto query to list all blob objects in a storage account and to list all read operations made per blob object.

To count the groups that have accessed a blob object, we analyze the role assignment that is included in every authorization log of the read operation. If the role assignment states a user group as the assignee we count it into the sum. If it states a single user as the assignee we count it into the sum as well, i.e. as a group with only one user. If the role assignment states a resource identifier as the assignee we identify its resource group association via the Azure REST API and count it as well.

This approach is limited by the fact that a user may have multiple role assignments (assigned directly or indirectly via user groups) which provide the necessary permission for the operation. Yet, we would argue that, first, in a cloud system

[3]https://docs.microsoft.com/en-us/azure/azure-monitor

[4]At the time of the writing it is not possible to post the logs directly to Azure Monitor; only a public preview is available, see https://docs.microsoft.com/en-us/azure/storage/common/monitor-storage

[5]as provided in the Azure documentation: https://azure.microsoft.com/en-us/blog/query-azure-storage-analytics-logs-in-azure-log-analytics/

[6]https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query

with a large number of users, creating role assignments for single users is not manageable and should be done via group assignments only. Second, we have discussed in Section IV that giving the same permission to a user via several groups is a smell in itself and is measured by the mixed duties smell.

3) Data lakes: For this metric we need to count the number of groups that have made write operations to a storage account, see (3). We do so analogously to the data broadcasts smell, with the difference that we retrieve logs of write operations rather than analyzing read operations.

B. Storage Configuration

The exposed data smell is detected using the three proposed metrics regarding encryption at rest (4), encryption in transit (5), and public accessibility (6). These are simple configuration options made on the storage account- or blob container-level and can be checked via the Azure Rest API[7].

C. Data Flow Tracking

The spaghetti flows metric counts the resource groups that have processed a data item, see (7). Azure provides Network Security Group (NSG) flow logs[8] which allow to log IP flows and other metadata. While this information can be used to analyze which general data flows exist between resource groups, information regarding single data items, e.g. identifiers of data items sent across the network, is not available.

For this paper, proposing a data flow tracking mechanism is out of scope. In a previous work we have shown that tracking data flows in cloud architectures is a complex task since the major cloud providers do not provide the necessary functionality [9]. Rather, users have to build complex systems to track blob objects, e.g. to track replications. Still, we have also identified a solution to how cloud providers may implement and offer such a tracking feature.

D. Role Assignments

In Section III-F and III-G we have proposed RBAC-related smells. While the mixed duties metric identifies the overlap of role assignments in user groups, the broad assignments metric identifies the scope depth of the role assignments. Role assignments can be retrieved via the Azure REST API, while user groups can be retrieved via the Microsoft Graph API[9].

We first map the role assignment tuple $(userGroup, permissions, resource)$ (see Section II) to Azure-specific entities, that is $(securityPrincipal, roleDefinition, resourceScope)$.

1) Mixed duties: To calculate this metric, see (8), we need to determine the tuple $(roleDefinition, resourceScope)$ of every role assignment given to a user group and compare them pairwise with all role assignments of all other groups.

In Azure, permissions are defined via *actions* and *notActions* that state an allowed or prohibited action in a specific service, e.g. a write operation in the Storage service. Actions can also include wildcards, e.g. *Microsoft.Storage/*/read*, to indicate read permission for all child resources within the Storage service. We compare two $(roleDefinition, resourceScope)$ tuples as follows:

First, check if there is an overlap between the scopes of tuple t_1 and tuple t_2, i.e. determine which resources are included in both scopes. We denote this set of resources with $overlapScope$. Second, build the set of effective actions $effAct_1$ and $effAct_2$ for every tuple which is calculated by subtracting a tuple's notActions from its actions (since notActions overwrite actions). Third, iterate through $effActions_1$ and check for every action if it is included in $effActions_2$. Fourth, build a tuple $(effAct_1 \cap effAct_2, overlapScope)$ that now includes the overlap of the two role assignments.

2) Broad assignments: To calculate this metric, see (9), we need to determine the role assignment's scope depth. In our implementation we use the abstraction levels *resource*, *resource group* and *subscription*, and simply assign them the values 1, 2 and 3 respectively. Note that in Azure, also more abstract levels, such as Management Groups, as well as more granular levels, such as blob containers, exist. The scope of a role assignment is specified as a path, e.g. a resource group is specified as */subscriptions/{subscriptionId}/resourceGroups/myresourcegroup1*.

E. Comparison to Other Cloud Providers

1) Resource groups and role assignments: In AWS, there is also the concept of resource groups. It is, however, designed as a means to manage resources in bulk, e.g. to automate tasks regarding a resource group. Role assignments in AWS cannot use a resource group as a scope. Rather, role assignments are defined as IAM policies which define a resource identifier as the scope. This identifier can include wildcards making it possible as well to identify a abstraction levels in this identifier.

GCP allows to build a hierarchy of resources and to attach policies to elements in the hierarchy. Resources further down in the hierarchy then inherit policies from resources higher up in the hierarchy. Similarly to Azure, each resource can only be part of one parent resource, making our RBAC metrics applicable to GCP as well.

2) Analyzing storage accesses: GCP offers a service called Cloud Audit Logs[10] that also provides the functionality to collect and analyze logs about storage accesses, including access types and authorization information. A similar service exists in AWS, called CloudTrail, that allows to log API requests to storage services[11].

3) Further differences and similarities: Obtaining information about encryption of data items and data sinks, as well as secure transfer in AWS and GCP is similar to Azure. This information can easily be retrieved via available REST APIs.

Performing data flow tracking of single data items is equally difficult in AWS and GCP as it is in Azure as we have described in our previous work [9].

[7] https://docs.microsoft.com/en-us/rest/api/storagerp

[8] https://docs.microsoft.com/en-us/azure/network-watcher/network-watcher-nsg-flow-logging-overview

[9] https://docs.microsoft.com/en-us/graph/api/resources/group

[10] https://cloud.google.com/storage/docs/audit-logs

[11] https://docs.aws.amazon.com/AmazonS3/latest/dev/cloudtrail-logging

VI. Related Work

To the best of the authors' knowledge, the notion of privacy smells has not been proposed before. Related work has investigated privacy problems in the cloud and in RBAC systems.

A. Privacy Strategies and Privacy in the Cloud

Bösch et al. [10] define *Privacy Dark Strategies* which they deduct from [2] as well. They build a framework to analyze these strategies and also integrate psychological factors. Yet, they do not translate their strategies to lower-level metrics.

Another related work by Jaeger et al. [11] proposes a semi-automated approach to collect measurement data from cloud systems regarding access control and data separation. They also propose concrete evaluation criteria for encryption, certificate quality and data separation. Yet, they measure data separation on the physical layer, i.e. separation of hardware used by different cloud users. As such, they do not propose privacy metrics regarding data flows or role assignments.

While we have proposed metrics that indicate *potential* privacy problems, other works have proposed privacy metrics that measure a level of privacy directly, e.g. in data sets. Eckhoff and Wagner [4] have reviewed over 80 such metrics.

B. Privacy in RBAC

He and Antón [12] developed a framework for representing privacy requirements in role policies. They do not, however, show how privacy may be measured in existing role policies.

Li et al. [13] also highlight the importance of implementing principles like least privilege and separation of duties in access control systems. Yet, they do not discuss concrete metrics to measure the degree to which these principles are fulfilled.

Our RBAC smells are also related to the area of role mining. Role mining assumes a given assignment of permissions to users and aims at defining an optimal set of roles that contains these permissions. Jafari et al. [14], for example, define *role cohesion* as a metric for how well a given set of user roles is defined, based on the permissions a user has needed in the past. In our mixed duties metric, however, we focus on the cohesion between user groups since we assume that similar duties are represented by a user group. Further metrics used in role mining include $\delta - consistency$ and *minimal noise* [15] which evaluate the minimality of role assignments rather than a privacy quality.

VII. Conclusions

In this paper we have proposed several metrics and privacy smells for cloud architectures. We have deducted these metrics from existing high-level strategies and have argued how smells may be detected based on these metrics. We have then shown how they can be measured on the technical level in Azure. While we have not proposed precise thresholds for the detection of smells, a cloud user can use custom expected thresholds for a cloud system to identify potential privacy problems. The metrics may also be used to monitor the respective privacy aspects of a cloud system over time. Additionally, auditors can make use of these metrics since they often only audit a fraction of a system; thus, they may use these indicators of potential problems to select a relevant sample of the system. Given the fact that privacy is often difficult to measure directly, we think that it is a valuable approach to measure indicators of *potential* problems using metrics and privacy smells based on common design strategies.

In future work we plan to identify further metrics and smells, and to identify respective mitigations. The proposed RBAC smells can be improved using machine learning techniques, e.g. using clustering techniques to allow a measurement of the distance between the permissions assigned to two user groups. We also plan to integrate the proposed metrics into our cloud monitoring tool Clouditor[12].

Acknowledgment

This work was partly funded by the Bavarian Ministry of Economic Affairs and Media, Energy and Technology, within the project Bayern-Cloud.

References

[1] Mina Deng, Kim Wuyts, Riccardo Scandariato, Bart Preneel, and Wouter Joosen. A privacy threat analysis framework: supporting the elicitation and fulfillment of privacy requirements. *Requirements Engineering*, 16(1):3–32, 2011.

[2] Jaap-Henk Hoepman. Privacy design strategies. In *IFIP International Information Security Conference*, pages 446–459. Springer, 2014.

[3] Eduardo B Fernandez, Nobukazu Yoshioka, and Hironori Washizaki. Patterns for security and privacy in cloud ecosystems. In *IEEE 2nd Workshop on Evolving Security and Privacy Requirements Engineering (ESPRE)*, pages 13–18, 2015.

[4] Isabel Wagner and David Eckhoff. Technical privacy metrics: a systematic survey. *ACM Computing Surveys (CSUR)*, 51(3):1–38, 2018.

[5] Martin Fowler. Refactoring: Improving the design of existing code. In *11th European Conference. Jyväskylä, Finland*, 1997.

[6] Francesca Arcelli Fontana, Vincenzo Ferme, Marco Zanoni, and Aiko Yamashita. Automatic metric thresholds derivation for code smell detection. In *IEEE/ACM 6th International Workshop on Emerging Trends in Software Metrics*, pages 44–53. IEEE, 2015.

[7] Vincent C Hu and Karen Ann Kent. *Guidelines for access control system evaluation metrics*. Citeseer, 2012.

[8] Michael Colesky, Jaap-Henk Hoepman, and Christiaan Hillen. A critical analysis of privacy design strategies. In *IEEE Security and Privacy Workshops (SPW)*, pages 33–40. IEEE, 2016.

[9] Immanuel Kunz, Valentina Casola, Angelika Schneider, Christian Banse, and Julian Schütte. Towards tracking data flows in cloud architectures. *arXiv preprint arXiv:2007.05212*, 2020.

[10] Christoph Bösch, Benjamin Erb, Frank Kargl, Henning Kopp, and Stefan Pfattheicher. Tales from the dark side: Privacy dark strategies and privacy dark patterns. *Proceedings on Privacy Enhancing Technologies*, 2016.

[11] Bernd Jaeger, Reiner Kraft, Sebastian Luhn, Annika Selzer, and Ulrich Waldmann. Access control and data separation metrics in cloud infrastructures. In *11th International Conference on Availability, Reliability and Security (ARES)*, pages 205–210. IEEE, 2016.

[12] Qingfeng He, Annie I Antón, et al. A framework for modeling privacy requirements in role engineering. In *Proc. of REFSQ*, volume 3, pages 137–146, 2003.

[13] Wei Li, Haishan Wan, Xunyi Ren, and Sheng Li. A refined rbac model for cloud computing. In *2012 IEEE/ACIS 11th International Conference on Computer and Information Science*, pages 43–48. IEEE, 2012.

[14] Mohammad Jafari, A Chinaei, Ken Barker, and Mohammad Fathian. Role mining in access history logs. *Journal of Information Assurance and Security*, 38, 2009.

[15] Jaideep Vaidya, Vijayalakshmi Atluri, and Qi Guo. The role mining problem: finding a minimal descriptive set of roles. In *Proceedings of the 12th ACM symposium on Access control models and technologies*, pages 175–184, 2007.

[12] https://github.com/clouditor/clouditor