Intelligent Reactive Access Control for Moving User Data

Yang Wang*, Armen Aghasaryan†, Arvind Shrihari*, David Pergament†, Guy-Bertrand Kamga†, Stéphane Betg-Brezetz†

*School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213
Email: [wang, arvinds]@cs.cmu.edu

†Service Infrastructure Research Domain
Alcatel-Lucent Bell Labs France
91620 Nozay, France
Email: name.surname@alcatel-lucent.com

Abstract—With the boom of social media, it has become increasingly easier for ordinary people to not only post their own content but share other people’s content on the Internet. In this paper, we conceptualize a growing problem of moving user data—once a user posts some content on the Internet, the data is largely out of her control; the content can be forwarded to or shared with other people, applications or websites, potentially causing various privacy issues. We present a technical solution that aims to provide users flexible fine-grained control over their moving data. Our system builds upon the ideas of data envelope with sticky policy, reactive access control, and privacy scores. Users can specify and enforce sticky policies of their data through our data envelope plug-ins. Our reactive access control mechanism allows users to grant access to their data on the fly, extending the pre-defined sticky policies to better fit with the dynamic nature of people’s sharing practices. Finally, the privacy score helps users make decisions about data requests by providing relevant privacy risk assessment information about the requesters.

I. INTRODUCTION

With the rise of user-generated content and social media, it becomes increasingly easier for ordinary people to post their own content and share other people’s content on the Internet. There is a trend that websites particularly social media sites such as Facebook and Twitter publish their Application Programming Interfaces (APIs). These open APIs allow third parties to develop applications that enable users to access their content on these sites and to share the content to other sites. We conceptualize this phenomena as moving user data—users’ data can be easily moved from one person to another, from one website to another, or from one communication channel (e.g., web) to another (e.g., email). Once a user generates some content on the Internet, the data is largely out of her control. The practices of moving user data can have considerable privacy implications. For instance, a recent study found over 4.4 million protected tweets (tweets that are only accessible by one’s followers on Twitter) were re-tweeted as public tweets, rendering these originally private tweets public and thus violating the privacy of the authors of the original tweets [1]. This telling example illustrates this growing challenge faced by Internet users how can they control their moving data on the Internet?

Our research aims to address this challenge. We present a framework that allows end users to control their moving data. Our framework builds upon the ideas of data envelopes with sticky policies, reactive access control and privacy scores. We use data envelopes as a means to encrypt and decrypt user data, and to allow data owners to specify a sticky policy containing access control rules that moves with the data. This sticky policy can be enforced as the data envelope moves [2].

Traditional access control requires users to make access control policies before data sharing. It has been shown that this ex-ante approach is too rigid to fit with the dynamic nature of people’s actual sharing behavior [3]. This is where reactive access control (RAC) comes to play. Under RAC, the data owner can answer incoming data requests as they come, creating a more flexible access control model [3].

While our approach can in principle apply to any moving data on the Internet across different application environments (such as email, social network sites, and blogs), we focus on email and social network service in this paper because they are most common among Internet activities.

Imagine a friend of friend wants to see a picture you posted on Facebook, should you grant this person to see your picture? You barely know this person. It would be useful if one can provide some relevant information about this requester to help you make the access control decisions “on the spot”. We designed a privacy score mechanism to provide that information. The privacy score is a quantitative measure of the extent to which allowing this requester to see this protected content would (or would not) compromise your privacy.

The main contributions of our work are: (1) flexible reactive access control policy management, (2) enforcement of access control policies across multiple application domains, and (3) decision support for end user access control decisions. In addition, we implemented a prototype system.

The rest of this paper is organized as follows. In section II...
we describe a concrete scenario that embodies the moving personal data problem. We then present our system design in section III, and prototype implementation in section IV. In section V, we discuss several challenges we are still facing and our future work. In section VI, we review related work and discuss how our approach differs and makes new contributions. Finally, we conclude in section VII.

II. Motivating Scenario

As a typical scenario, we consider an enterprise where employees can communicate and exchange information using an internal professional social network such as Jive Engage\(^1\). Alice, an employee of the company, posts a status update on Jive Engage. Bob, a friend of Alice on this social network, receives an email containing Alice’s original post, and he forwards the email to Calvin who does not work for their company. Alice barely knows Calvin. If Alice’s original post was plain text, then Calvin will see it in the forwarded email. This would be a privacy leakage. If the forwarded email only contains a link to Alice’s post which she only allowed her friends on Jive to see, then Calvin would not be able to see it. We want to avoid having to choose between these two extreme cases. Why not allow Calvin to ask Alice’s permission to this post? Can we help Alice decide whether to approve such a request or not? Our solution aims to address these questions.

III. Intelligent Reactive Access Control

A. Overall Design

Our system is comprised of three major components: data envelope plug-ins, a reactive access control (RAC) service, and a privacy reputation score service. Figure 1 shows a high level overview of the system architecture. In order to enforce access controls on users’ moving data, we assume that Alice uses a Privacy Data Envelope (PDE) plug-in installed in her web browser when she makes the post on Jive Engage. Alice specifies a policy of this post, that is who can see her post (on Jive Engage or in other environments). This policy is associated with the post and the set is encrypted by the plug-in to generate the Privacy Data Envelope (PDE). The post is then protected by the PDE plug-in so that only people who were allowed to see it as specified in the policy can see the post. In order to actually view the post, these intended recipients need to install the PDE plug-in and authenticate themselves, and then the plug-in will decrypt and display the post. If Calvin tries to access Alice’s post on Jive, he will get a warning that asks him to install the PDE plug-in first. After installation, the plug-in checks the policy associated with Alice’s post and finds that Calvin is not among the ones who were allowed to see Alice’s post. Therefore the plug-in would not decrypt the post. The same situation would happen if Calvin gets the post through an e-mail forwarded by another user, Bob, having an authorized access to the Alice’s post on Jive. In this case, the appropriate PDE plug-in would be installed on Calvin’s email client.

If Calvin wants to see Alice’s post, he can use his PDE plug-in to request access to the post. The plug-in then relays the request to a remote RAC service. In order to make and reply to reactive access control requests, a user needs to authenticate himself on the RAC service either by having an account on the RAC service directly or logging in using external identity mechanisms such as Facebook or OAuth. Once the RAC service receives Calvin’s request, it will route that request to Alice, e.g., via email. Alice gets an email notification of the request and logs in the RAC service to view the request. On the request page, Alice will see that the requester Calvin asked her permission to see that post. Since Alice knows little about Calvin, the Privacy Score service will automatically compute a score of privacy risk if Alice would approve this request. The score will be presented on the request page to help Alice make the decision. Alice’s decision will be sent back to Calvin’s PDE plug-in. If it is an approval, then the PDE plug-in will decrypt the post and show it to Calvin. If Alice rejects or ignores the request, she can choose to have her PDE plug-in send “your request has not been approved yet” rather than “your request has been denied”. The former supports a form of built-in plausible deniability [4]. Imagine that Alice and Calvin are friends, but Alice does not want Calvin to see that particular post. This vague answer would help avoid potential tension between Alice and Calvin as a result of Alice’s denial.

B. Privacy Data Envelope

We rely therefore on the Privacy Data Envelope approach situeted in the family of sticky policies that allows dealing with hierarchical data structures as well as covering group privacy protection scenarios [2]. In this approach each piece of information identified as privacy-sensitive is embodied (or “enveloped”) into a data structure that, in addition to the initial raw data, carries privacy-related properties and policies. The PDE structure contains three fields:

- Privacy-sensitive entities (or sensitive entities): define the parts of raw data that are different from the point of view of privacy, e.g. user profile data, user-generated

\(^1\)www.jivesoftware.com

![Fig. 1. Overall design of the system](image-url)
content, or user-system interaction traces; these parts are characterized by their individual properties.

- Properties (or metadata): specify some semantics of the respective data entities, e.g. the data type and category, owner, or issue date.
- Privacy policies: indicate what actions are authorized when the PDE travels across the network, (e.g., access control, time-to-live, data handling and disclosure policies) and what obligations must be fulfilled (e.g., data deletion after a certain time period, notification or consent request to owners). These policies are enforced locally at each PDE recipient’s client location.

Each PDE is currently encrypted by using a simple encryption schema based on a symmetric key. The PDEs travel across the network in an encrypted form, therefore each node of the network (we refer to it as the PDE network) that needs to access to its content has to possess the symmetric key. In the basic version, we suppose that this key is unique and available to all the certified nodes of the network. In order to become a node of the PDE network and to have potential access to PDE contents, an end-user application (in a browser or email client) needs to integrate the previously mentioned PDE plug-in. This plug-in is in charge of decrypting the PDE and enforcing its policies in the specific application environment.

More sophisticated security schemes like Public Key Infrastructure or hybrid approaches can be applied on the top of this basic mechanism. This approach requires the policies to be specified using the application specific user identifiers which will be subsequently verified by the recipient’s PDE plug-in integrated in the respective application environment. Finally, the PDE approach is integrated with the reactive access control service by means of specific policies which return an obligation indicating that for the given content item consent of the owner must be obtained. The obligation returned by the policy decision point carries the RAC request signature. The obligation is then enforced by the PDE plug-in. The final request includes the requester id and some content metadata.

C. Reactive Access Control (RAC) Service

The RAC service helps manage requests made on the fly for the content owner’s data. We design the RAC service as a stand-alone web service that can be hosted on a remote server. Whenever a requester asks for permission to a Privacy Data Envelope, that request is sent to the RAC Server used by the owner of the data envelope. The server notifies the owner of the request and once the owner responds to the request, the RAC service can respond to the requester’s PDE plug-in with the owner’s response. Anyone can start a RAC server and when a user creates a data envelope, he or she must include the location of his or her RAC server so requests for the data can be routed properly.

The distributed design of the RAC service fits the distributed nature of the requesters and content-owners. Similar to the design of servers in the Diaspora project, a single user or a group of users could run their own RAC servers. For example, a family could maintain a RAC server for requests for family photos that have been put in data envelopes.

When the RAC service receives a data request, it identifies the requester and the data owner. It then prepares a page for the data owner to view the request. It sends an email to the data owner with the URL of the request page hosted on the RAC server. On the request page, the data owner can see who the requester was and what data is being requested. The owner is presented with several options including approval, denial, or ignore of the request. The owner can optionally include a text explanation of her decision. If it is an approval, the owner can also specify obligations as part of the approval, for instance, the expiration date of the approval (i.e., after certain period of time, the data envelope can no longer be decrypted).

D. Privacy Score

In order to assist users in making decisions about data requests, we designed the Friends-Oriented Reputation Privacy Score (FORPS). The basic idea of the FORPS mechanism, consists in taking advantage of the overall knowledge present in a social network and that is accessible to a given user (e.g., Alice) [5]. Then, the system tries to estimate the danger that another user (e.g., Calvin) may represent with respect to a non-desirable propagation of Alice’s sensitive data. This can be done by aggregating different sources of information characterizing Calvin’s profile and behavior:

- public profiles of other users available in the social network,
- the private profile of Calvin, and more importantly,
- the information that the friends of Alice possesses or have access to, concerning Calvin, e.g. likes or comments that Calvin leaves on photos belonging to one of the friends of Alice; this includes also the Calvin’s privacy score computed by the friends of Alice.

The FORPS system allows Alice to define her privacy sensitivity profile which is characterized by the themes/categories, the object-types that are relevant for Alice. For instance, Alice may want only some of her content items concerning a specific topic (e.g. family) to stay in a restricted area of users, other topics can be propagated. The same applies to different object types such as posts, photos, videos, etc. These preferences are taken into account by the system to calculate different privacy reputation scores of Calvin per theme and object type and then to obtain an aggregated score. The score computation is based on different behavioral factors characterizing information propagation in social networks, e.g. propagator propensity, information sensitivity, and user popularity.

IV. Prototype Implementation

We implemented a proof-of-concept prototype to test the feasibility of our proposed approach based on the Privacy Data Envelope (PDE) plug-ins, the Reactive Access Control (RAC) service, and the Friends-Oriented Reputation Privacy Score (FORPS) mechanism.

2https://joindiaspora.com/
A. Communication Protocol

Our system was designed to work in a distributed fashion. While the PDE plug-ins will need to be installed on end user machines (integrated within the application clients), the RAC service and Privacy Score service can be hosted on different remote servers. We designed and implemented a communication protocol for the three major components in our distributed system to communicate with each other. Figure 2 depicts the overall design of the communication protocol.

1) Plug-in Requests for Data: When Calvin requests to see a piece of Alice’s data through his PDE plug-in, the plug-in makes a HTTP POST to the appropriate RAC server URL passing a JSON (Javascript Object Notation) object encrypted with the plug-in key:

```
{action_name : ‘read’, ‘forward’, or ‘save’
content_name : content title,
content_guid : sha2 hash of content,
content_metadata : content description,
content_image : Base64(jpeg image),
owner_guid : owner guid,
requester_guid: requester guid}
```

If the request is successfully created, the RAC server responds with a message of “request created” and a unique request id; otherwise it returns “request not created”.

2) Plug-in Pings for Request Status: In order to get a response from the RAC server, Calvin’s plug-in makes period-ical HTTP GET requests to the appropriate RAC server URL passing the following parameter: status_id: unique request id encrypted with plug-in key.

The RAC server then responds with a request status code which is an integer that can take a value of -1, 0, 1, or 2. The code indicates the request status: “-1” for “request not found”, “0” for “request pending”, “1” for “request approved”, and “2” for “request rejected”. “Request pending” means the data owner has not decided whether to approve or reject the request. The rate at which the plug-in pings the RAC server can be configured and it will decay over time. For instance, within the 1st hour of the request, the plug-in pings every 5 seconds, and during the 2nd hour it pings every 10 seconds, and so forth.

B. PDE Plug-in

The PDE plug-ins are conceived as extensions or add-ins of existing communication tools. We focused our development on two such tools, a web browser and an email client, and implemented two instances of PDE plug-ins: one as a Mozilla Firefox extension and the other as a Microsoft Outlook add-in. The Firefox extension is based on XPCOM (Cross Platform Component Object Model) which allows developers to add new functionalities and to customize Mozilla applications. The Outlook add-in is a Microsoft COM (Component Object Model) add-in that has access to the Outlook’s object model that is necessary to control different events, e.g., intercepting end-user’s actions, or checking if an action is subject to a policy. Each PDE plug-in is composed of the following main elements [2]:

- **Contact Matrix** for storing a list of user contacts from different application environments (e.g. Jive Engage, Facebook, or email); each row of this matrix corresponds to a user identifier and each column corresponds to a communication environment;
- **Application Environment Access Manager** for populating the Contact Matrix to identify the current user within different application environments; the user identification is done by listening to an open session with an application (e.g. Jive Engage, Facebook, or email);
- **Policy Enforcement Point (PEP)** for sending requests to the Policy Decision Point (PDP) and enforcing the resulting decisions and obligations;
- **PDP** for evaluating the PEP requests according to the associated policies;
- **PDE Builder** for creating a Privacy Data Envelope (PDE), i.e. binding a content with the related policies and applying the encryption schema;
- **PDE Detector** for detecting PDE entities and triggering the PEP.

In the implementation of the PDE plug-in, we modified the PEP element to support reactive access control policies. To achieve this, a user can create a PDE with policies containing a specific obligation which requires to contact another user (through the RAC Service) to confirm or invalidate the PDP decision. In this situation, the PDP decision is an intermediate decision and the final decision is the one provided by the RAC Service that can be considered as a distant PDP.

For instance, considering the motivating scenario described in Section II, Alice can associate a policy to her post, stating that she can be contacted through the RAC Service about requests to access her post by anyone who is not her friend on Jive Engage. When Calvin tries to access Alice’s post (either on Jive Engage or on his Outlook email client), the PDE plug-in evaluates the policy and discovers that Calvin is not authorized to view the post, but instead an authorization request can be sent to Alice using the RAC service according to the policy specified by Alice. Note that these policies can
narrow down the circle of individuals authorized to make RAC requests, e.g., only “friends of friends” in the social network.

C. RAC Service

The RAC server has been implemented using the Ruby on Rails web development framework. It serves two primary functions: allowing content owners to respond to requests for their content and providing an API for PDE plug-ins to create and inquire about requests.

Content owners can create an account on the server directly or connect through their Facebook accounts. Authentication through Facebook has been implemented as specified in Facebook documentation\(^3\). The RAC server stores a user’s first name, last name, and an email address. If the user has an account directly on the RAC server and not through Facebook, a hash of the user’s password is also stored. The server also provides a page for users to view and manage their requests.

The API for PDE plug-ins consists of two interfaces that are intended only for access by the plug-ins. The “requests” interface creates a request and replies with a unique request id in plaintext when sent a HTTP POST with request information. The “status” interface replies with plaintext information about a request when sent a HTTP GET with a request id. For both of these pages, the data passed is decrypted on the backend before processing. The server uses a MySQL database to store user and request information using a normalized data model.

D. Privacy Score

The FORPS system [5] is implemented as a Facebook application by using ActionScript\(^4\) and Adobe Flex\(^5\). From the perspective of the RAC service, the FORPS is simply embedded in an iFrame. We privilege this approach to provide the data owner the possibility to tune the computation process and the visualization at his convenience. Since this module is implemented as a Facebook application, both the data owner and requester need to have a Facebook account in order to compute the privacy reputation score. Once the data owner logs in the Facebook application, the Facebook identifier of the requester will be automatically fed from the RAC service to the application. Before triggering of the computation, the data owner can orient the computation process according to some specific categories, e.g., the themes related to the semantic category of the data being requested (e.g., family, politics, professional), or their object type (e.g., posts, photos, videos).

The results of the privacy score computation are displayed on the interface illustrated in Figure 3. The interface presents to the data owner a global privacy score of the data requester, as well as some specific privacy scores, ordered according to his preferences expressed in terms of most relevant object types and themes; these preferences are stored in the privacy sensitivity profile of the data owner. In addition, the data owner can retrieve more explanation on the meaning of each privacy score by clicking on the score, and the underlying “raw data” used to compute the score such as photos or status updates will appear. Some of these data belong to the requester, while other data may belong to common friends of the requester and the owner where the requester has added some information, e.g., commenting a post by a common friend.

As Figure 4 shows, the privacy score application is displayed in the request page using an iFrame. The data owner can log in the privacy score application and select the requester, and the corresponding privacy score information can be computed and displayed.

V. DISCUSSION

A. Distribution of RAC Service

The distributed design of the system fits with the distributed nature of the requesters and content-owners. We proposed a design that allows content-owners to specify any RAC server including, perhaps, their own, for managing their request. The distributed design of the RAC service also provides flexibility in the RAC server implementation itself. As long as a server follows the global communication protocols, it can function as it wishes internally. For example, the RAC server handling requests for a family’s photos could have an internal approval system where children’s responses to requests need to be approved by parents. This flexibility in implementation also

\(^3\)http://developers.facebook.com/docs/authentication
\(^4\)http://code.google.com/p/facebook-actionscript-api
\(^5\)http://www.adobe.com/devnet/facebook.html
allows for third-parties to create custom RAC or host RAC services for the needs of different users.

B. Identity Management

A challenge in implementing an effective integration of the plug-ins and the RAC service across the entire web is identifying and authenticating users. Whereas identity can be managed more easily in closed, internal systems like corporate networks, the lack of central control on the web poses challenges. For example, in a solution that uses Facebook to identify requesters to content-owners, there is a threat of requesters impersonating friends of content-owners to gain access to data. This impersonation could be done easily on Facebook by changing one’s name and the photos shown on the profile page to match those of a content owner’s friend. When the content owner receives the deceptive request, a cursory visit to the requester’s Facebook profile of the request might convince the content-owner that the requester is indeed a friend. If the content-owner is not on Facebook, it becomes even easier to be deceived by simply the name of the requester. While the privacy score can help avoid such situations, it cannot be relied on to always identify deceptive requests.

C. Plug-in Security

Another challenge is to ensure the trustworthiness of requests from a PDE plug-in. If there was no way to verify the validity of a request for access to content, a rogue plug-in could make requests on behalf of individuals the content-owner trusts. One solution to this problem is to authenticate the validity of a request by having the plug-in encrypt requests using the same key used to generate the data-envelope. RAC servers can decrypt this message using the key. This poses the challenge of keeping the keys secret. This is especially difficult in a fully decentralized system where RAC servers can be set up by anyone.

D. Scalability

There are a number of design decisions in the system that will impact its scalability. The most appropriate system architecture can vary significantly based on requirements and constraints of the environment.

1) Frequency of Status Requests: When a content owner responds to a request, this response must be communicated back to the requester. This response can be made in a ‘push’ manner where the requester’s plug-in simply waits for a message from the RAC server. In an environment where a “push” is feasible, it could work well. Many networks including the Internet, however, are not well suited for a “push” architecture and instead will need to implement a “pull” architecture. In this scenario, a plug-in can make status requests to the server and the server can respond with information in the response if that information is available. With a “pull” architecture, the policy determining the frequency of requests is crucial to a scalable system.

The frequency of status requests from the plug-in will depend on the requirements of the environment. For example, in a closed system where the RAC server does not get frequent traffic, the server can allocate more resources to handling frequent status requests from plug-ins.

In an environment where RAC server resources are scarce, it might be more appropriate to have infrequent status requests where the interval between requests increase as time passes and after a certain amount of time has passed, the status request times out. For timed-out requests, if a requester actively inquires about the status on a request, the plug-in can send a new status request to the server. For an Internet-wide system where RAC servers could have varying constraints and predicting request patterns is difficult, more conservative schemes would be better. Another solution for an Internet-wide system would be to have the RAC server specify a policy for the frequency of requests at the time of a request creation when the plug-in first contacts the server.

2) Amount of Content Metadata Sent: Another factor that affects the scalability of the system is the amount of metadata that is sent to the RAC server with each data request. Some metadata, such as a unique content id, must be sent to the server so that the content is identifiable to the user. Additional metadata like a sample of the content, size information, and a title for the content can be sent. As more metadata is sent the network load increases but it becomes easier for the content-owner to recognize the content and evaluate the request. This trade-off and requirements on the particular system need to be considered when making this design decision.

Another solution to this problem is to not have to send the content metadata with each data request and instead to inform the RAC server of the creation of a data envelope and provide some metadata at the time of creation. Thus, future references to the content can be done simply through a global id or hash that identifies the content. This solution requires the RAC server to store significantly more data about users’ content but allows data-envelopes as well as requests for access to be smaller as they would contain no metadata outside of the content identifier.

Similarly, the plug-in itself (and not the RAC server) could maintain the content metadata at the time of creation of a data envelope. As long as the plug-in can match a content id to the stored metadata, it can provide the owner with this metadata when the RAC server displays the content-id. This solution requires the plug-in to be active when the content owner is accessing the RAC server and greatly increases storage load on the plug-in.

3) Performance Evaluations: As a first step, we have performed some preliminary evaluations on the data envelope plug-in mechanisms [2]. We tested the content published on the Jive Engage social network via our Firefox PDE plug-in. We measured the processing time for the plug-in to parse the content, decrypt the content, extract the sticky policy, make the access granting decision, and display the plain text. We found that the plug-in took less than one second on average to process a 5-mega-byte web page. The performance of the whole integrated system still needs to be evaluated.
E. Applicability of Privacy Score

Our current implementation of the privacy score mechanism, the FORPS system [5] is geared towards online social networks. The privacy score is computed using information retrieved from a social network, but the result can then be applied to other communication channels helping the user decide, for example, whether an access to an email message should be granted or not to a given requester.

In addition, if the data owner knows or the data requester provides (e.g., the RAC service) the various services the data requester uses (and better yet her account names), we can retrieve public information available about the requester from these services and incorporating that information into the privacy score computation.

F. Usability

Our system would fail miserably if users find it difficult to use, or its usage compromises rather than facilitate their current sharing practices. One open question is who should be allowed to send data owner requests. The reactive access control mechanism is meant to enable more flexible data sharing. However, we do not want data owners to be overwhelmed with incoming requests. Whether allowing anyone including complete strangers to make data requests is debatable and deserves further exploration. In future version of our implementation, we could support settings in the plug-ins that allow data owner to specify who can send them data requests such as “nobody”, “contacts/friends”, “friends of friends”; and “everyone”. Another related issue is how frequent data owners will receive data requests.

Overly frequent notifications from the RAC service about data requests can be overwhelming and thus detrimental to the user experience of the content-owner. Depending on the type of content and external factors such as the exposure it gets, the content-owner could get an unmanageable number of requests. This problem can be mitigated by allowing flexible notification settings. This flexibility could involve allowing the user to specify the frequency of request notifications. Content owners could also set up filters that determine which requests can be ignored based on what content is being requested or who the requester is. These ideas can be explored with user studies.

VI. RELATED WORK

In this section we first review related work in each of the three major building blocks of our system: sticky policy, reactive access control, and privacy score. We then review literature in protecting user privacy in social media in general.

A. Sticky Policy

To deal with the privacy control of moving data, a family of approaches based on sticky policies has been proposed. The basic idea consists in accompanying the moving data with privacy protection policies which should apply all along the movement path of these data [6], [7]. Some approaches also allow the use of multiple policy languages as well as protect data shared by multiple applications [8]. They rely nevertheless on a dedicated user identification service and require a trusted authority which we want to avoid as a source of usability overhead. The system presented in this paper relies on the approach of Privacy Data Envelopes (PDE) which proposes a policy-based privacy technology to protect data across different communication and Web 2.0 applications [2]. This technology is based on PDE plug-ins having a dual role of data encryption/decryption and policy enforcement. In contrast with the existing solutions, the PDE Plug-in is a lightweight software add-on fully distributed and embedded in different application environments, directly using the existing specific application features and their user identification schemes.

B. Reactive Access Control

Traditional ex-ante access control mechanisms have been shown to be rigid and somewhat disconnected with the actual access control behaviors. Palen and Dourish argued that personal privacy management is a dynamic and dialectic process in which people make sharing decisions contingent upon the constraints at the moment of sharing rather than enforcement of pre-specified rules [9]. Drawing from empirical investigations of access control behavior, Stevens and Wulf show that using ex-ante access control alone can be insufficient and suggest an array of additional access control mechanisms, reactive access control (RAC) being one of them [3]. Under RAC, the data owner can answer incoming data requests as they come, complementing ex-ante access control in creating a more flexible access control model.

The idea of RAC starts to gain attention in academia and industry. Bauer et al. implemented a smart-phone, RAC-based system to control access to physical resources (e.g., offices) and showed its use in practice [10]. Mazurek et al. investigated the use of reactive access control using a experience sampling method [11]. A plug-in called airdropper[6] for the online storage site Dropbox allows its users to request access to data owned by other users. Despite these prior work, to our best knowledge, our RAC system could be the first that provides general-purpose reactive access control as a stand-alone and distributed service.

C. Privacy Score

Various systems proposed the concept of privacy score which can be used to alert users about the visibility and protection of their sensitive data. They are implemented as websites (e.g., Profile Watch 7) or as Facebook applications (e.g., ‘Privacy Check’ 8). Liu and Terzi proposed a privacy score on social networking sites (SNS). The scores are computed by considering two factors, the visibility and the sensitivity of the user’s data [12]. Our privacy reputation score differs from the aforementioned approaches in that it takes different input data and uses a different algorithmic approach for the score computation [5]. Instead of analyzing

6http://www.airdropper.com/
7http://www.profilewatch.org/
8http://www.rabidgremlin.com/fbprivacy/
only the data owner’s private or public data, our approach also considers the particular usage context defined by another user (the data requester) who is requesting an access to the data owner’s information. This request can be formulated either as a friendship request in a social network or any other request to access a specific content item of the data owner. The score represents the estimated privacy risk to the data owner if the request is granted.

D. Protecting User Privacy in Social Media

With the phenomenal rise of social media, privacy issues in social media, particularly in social network sites (SNSs) such as Facebook, only seem to get more intense. Various privacy risks have been identified in the literature and a growing body of work aims to address these privacy issues in social media. Most systems include some flavor of encryption to protect user data and use access control on top of the encryption. Lucas and Borisov built flyByNight that encrypts the SNS user information using a public key infrastructure [13]. Similarly, Guha et al. developed NOYB which also encrypt users’ SNS profile information so that only authorized parties can view the data. In their scheme, users’ sensitive data items are pseudo-randomly substituted by another user’s data item. One benefit is that it is more difficult for SNS operators to detect the use of NOYB [14]. Baden et al. developed Persona, which uses attribute-based encryption to protect user data [15]. Sun et al. used role-based encryption to hide user data [16]. Masoumzadeh and Joshi designed an ontology-based access control model of SNS data [17].

In addition to encryption, distribution of data is another privacy enhancing technique. Systems such as Lockr [18] and SPac [19] resemble Diaspora, the high-profile open source alternative to Facebook. These distributed systems enable users to store their SNS information locally or on a trusted third-party server rather than the centralized server operated by SNS owners such as Facebook. More recently, Fischer et al. proposed using email as a federated platform of various SNS services and a data storage of these SNS data.

E. Summary

Our framework relies on encryption, sticky policy, distribution, flexible reactive access control, and decision support for user access control. Our PDE mechanism proposes a policy-based privacy technology to protect data across different communication and Web 2.0 applications. Moreover, this technology is based on an immersive Plug-in distributed and embedded in a “best effort” way within the different applicative environments (i.e., directly using the application features and their user identification systems). As with Lockr and Diaspora, our framework is de-centralized, allowing users to store their information on their choice of trusted third-party servers. Users can easily move their data from one source to the other. Like most existing solutions, our framework encrypts the user data. In fact, our framework is fairly opaque to encryption. Designers can easily plug-in their choice of encryption scheme. However, our current implementation does not include a key management component but can be built or customized accordingly. Most existing solutions also supports access control, but none of them supports reactive access control, which we believe is more flexible and fits better with people’s actual sharing practices.

We found little previous work that attempts to integrate all these complementary privacy enhancing techniques together. Taking our system as a whole, it provides a more comprehensive list of privacy features that address various privacy or trust issues.

VII. Conclusion

As user-generated content continues to grow on the Internet, the problem of moving personal data will get more intense. Our technical approach aims to provide a comprehensive and integrated solution to help end users reclaim their well-deserved control over their own data, even after the data has gone out to the Internet. In future work, we plan to run user evaluations to empirically assess the usability and effectiveness of our system. We also plan to explore ways to learn from a content-owner’s previous decisions when responding to requests. We can use this knowledge to try and make it easier for the content-owner to deal with future requests by, for example, setting defaults that match patterns of behavior the user might have shown in the past.

ACKNOWLEDGMENT

We gratefully acknowledge Lorrie Faith Cranor, Kami Vania, Rich Shay, and Saranga Komanduri from the CMU Usable Privacy and Security (CUPS) Lab as well as Mahmoud Ghorbel, Marie-Pascale Dupont, and Sophie Piekarcz from the Alcatel-Lucent Bell Labs for their support and advice. This work is supported by CMU CyLab and Alcatel-Lucent Bell Labs under a research collaboration grant.

REFERENCES


