Paranoid: Privacy through Granular Separation of User Identities
Project Proposal

Brandon Tan (tjiansin)
Brown University

Irvin Lim (ilim5)
Brown University

1 Introduction

Paranoid explores a potential approach that separates user's identity and sensitive information from third party offerings and services. *Paranoid* is a client-side, peer-to-peer solution that places explicit control of user's private data into their own hands. This is achieved by storing private data completely on users' local machines, substituting private data values in untrusted services' databases with placeholders, DOM manipulation in the browser to display private data in the client side, as well as a private data management tool for sharing and explicitly granting permissions to another user to access data.

1.1 Goals of the Project

In the recent years, there have been an increasing number of high profile instances of users' personal information being compromised. This includes, but is not limited to:

1. Data breaches arising from inadequate access control, e.g. Equifax announced a data breach in 2017 that exposed the personal information of over 140 million consumers worldwide [1];

2. Governments and law enforcement agencies that could potentially demand private corporations to hand over its users' personal information, e.g. the Federal Bureau of Investigation (FBI) in the United States wanted Apple to sign its software that would allow FBI to unlock an iPhone recovered in a shooting case in San Bernadino, California in 2015;

3. Lack of consent for services to pass on users' personal information to external vendors, e.g. the Facebook/Cambridge Analytica scandal which resulted in 87 million users' Facebook profiles being acquired by Cambridge Analytica for voter targeting in the 2016 US election;

4. Lack of guarantee for users' personal information being stored and processed securely by the service;

5. Personal data acquired from multiple sources can be correlated to paint a comprehensive overview of a users' online identity.

There has been various approaches in solving the above mentioned challenges, including information flow control (IFC) frameworks such as DStar, Resin and Jacqueline, which require implicit trust in the service itself to adequately implement these guarantees. Additionally, while services like Facebook is starting to provide a fine-grained control of defining specific sets of users that should be allowed access to data, it is complex to implement and potentially buggy [3]. This poses a practical problem for end users who wish to take proactive measures to start protecting their personal information, but yet still want to make use of existing services today.

With that in mind, we propose a solution to store all data that is deemed as sensitive completely in client-side storage, and will not be transmitted to the third-party service in any way unless granted by the user, while aiming to maintain as much existing functionality as possible without divulging said information to the service. This way, any data breaches or unexpected usage of the data stored in the service will not affect users who have made use of *Paranoid*.

1.2 Targeted Services

In the point of view of *Paranoid*, service providers can be classified into two distinct categories: *data platforms* and *data consumers*. For example, a data consumer is a service like Equifax or an online banking website, which requires all data values, usually personal identifiable information (PII), submitted to the service to be true and accurate, as otherwise it would fail to perform a critical function to serve the user and would be of no value. On the other hand, a data platform is something like Facebook or Twitter, which could continue

to function well for the user, even if the user uses a fake name for their profile, for example.

For the purposes of this paper, we will be focusing mainly on data platforms as they allow us to assume a stricter threat model. However, we will also be discussing how *Paranoid* would work in the case of data consumers, albeit providing lesser privacy guarantees.

1.3 Security Overview

1.3.1 Actors

Paranoid defines a user that has been provided access to private data as a "data observer" of the data, with "data owner" being the user who it belongs to. Third party companies or entities that provide services to users are defined as "service providers".

1.3.2 Threat Model

In all cases, we assume that the data owner takes proper and reasonable care to protect their data. We see that the threat model would differ for each class of service:

- Data platforms assume a stricter threat model where service providers are deemed untrustworthy and should not have control or access to users' private data. We further assume data observers as being trustworthy and will not divulge private data that they have been granted access to. This thus warrants that data should not be sent to service providers in any way.
- Data consumers assume a similar threat model, but because they require private data to function, *Paranoid* cannot provide technical guarantees of private data once sent to the service provider. As such, for the case of data consumers, we extend our assumption of data consumers, as well as any other service providers that they send data to, to be trustworthy. In that way, a service provider can also be a data observer, if the data owner has granted access.

2 Proposed Design

To address the goals and concerns listed above, we propose a system that gives users explicit control over their personal data while retaining majority functionality of modern-day web applications with the use of placeholders.

2.1 Using Placeholders

When a user registers for an account on a web service, instead of divulging personal information such as full name, birthdate, home address/city, etc., a *Paranoid*-compatible service would assign the user a unique random identifier UID, that is stored in the service's database, and is returned to the user. No private data will be sent from the user to the service, as it is not granted.

For example, the user's Facebook profile would now display placeholders for certain fields using a tuple $< UID, FieldName >$, in place of the user's name. Only the user themselves, as well as select users that have been explicitly given permission, will be able to view the actual, "unmasked" information on any webpage from the service that displays the user's personal data (or in this case, the placeholder). This is a two step process:

1. When a web service returns any text content via HTTP, such as HTML or JSON, it could potentially contain user data placeholders to be substituted. For example, instead of Facebook displaying "John Smith" in the name field of a profile page, `PARANOIDPLACEHOLDER-U21VZ3A0hNYRC5H9-name` could be displayed instead.

2. If a user has been granted permission to view this specific key `U21VZ3A0hNYRC5H9-name`, the actual data value would be fetched and substituted into the text content by the data observer's browser. This reconstruction is done by rendering a copy of the original DOM with the placeholders, and replacing them with the actual data values. Otherwise, if no access is granted, the placeholders will remain as they are.

These fields can be a well-known list, such as name, email address, country, birthday, etc., but also can possibly be user-defined by the data owner. This may require additional modification to the service to support it, but a simple implementation would be to just for the data owner to submit placeholders in the form field and store the relevant mapping in his/her storage, but would be limited to just strings and text-based data formats that is sufficiently long enough to store the full key name.

2.2 Granting Permission

If we use the example of John Smith's Facebook page as per before, to everyone else his profile would look anonymous. However, John could choose to explicitly share his identify on these services with select individuals. We propose a client-side tool, the *ID Manager* that performs access control management between users of individual data keys owned by the user.

In order for data owner U_1 to allow data observer U_2 access to a certain field on a service, the ID Manager for U_1 has to transmit mappings of $<< UID, FieldName >, Data >$ to U_2. This way, U_2 will be able to see U_1's Facebook profile page reconciled with the pieces of information that U_1 chose to share explicitly with U_2.

We will not be discussing the transport mechanism in which private data is sent between users, but we assume that it is reasonably secure (ensuring confidentiality) and cannot be subjected to tampering (ensuring integrity). For example, we plan to explore using Keybase for storing and transporting of keys and data between users, but a homebrew solution written from scratch could be explored as well.

In the case of Keybase, it provides a couple of neat features such as end-to-end encryption as well as automatic data syncing to all receivers upon file updates. These features provides a mechanism for *Paranoid* to push and update data, which can be useful if a the data owner decides to update some private data. Keybase can help facilitate the propagation of these updates without needing the data observer to explicitly accept the changes. Likewise, a data owner can also rescind the any data sent by propagating a file delete action (assuming data observer does not make a local copy).

2.3 Customized Browser

We find that just replacing placeholders is insufficient. JavaScript code originating from the service, which is executed on the client side, can still execute after reconstruction is completed. This opens up an avenue for adversaries to siphon out private data, potentially by sites which may want to retrieve the actual personal data that is rendered on the user's browser. To address this issue, *Paranoid* provides a modified web browser, such that the JavaScript engine is only allowed to execute and make modifications to the original copy of the DOM (prior to the placeholders being replaced). Hence, whenever a modification to the original DOM occurs, possibly tracked using event listeners like `MutationObserver`, these DOM changes will be propagated to the new DOM, which does not break existing client-side interaction on the browser. Similarly, user interactions with the new DOM also have to be propagated back to the original DOM in which untrusted JavaScript can execute, possibly by simply proxying `onClick`, etc. event listeners from the currently rendered DOM to the original DOM as well.

2.4 Storage Mechanism

All private data should be stored locally on the data owner's filesystem, and only shared with other users via a secure channel using the ID Manager when explicitly granted. We can use a simple key-value store, such as Redis or even using individual files as keys, to store a key-value mapping as follows:

$$< Domain, UID, FieldName > \rightarrow DataValue$$

We also need to store granted permissions, so that the data owner can keep track of which users currently have been given prior access to the data.

Note that if the ID Manager for the data owner is not currently available, data cannot be transferred.

3 Next Steps

We will be exploring the feasibility of this idea, verifying any edge cases that may arise that may compromise the confidentiality or integrity of the private data, coming up with a prototype and benchmarking its performance using real-world applications.

We will also be finding out the limitations of such an idea, if any, as well as whether such a solution would be sufficient to make a balance between user-friendliness and granular control over user private data.

References

[1] O'Brien, S. (2017). Giant Equifax data breach: 143 million people could be affected. *CNN*. Retrieved from https://money.cnn.com/2017/09/07/technology/business/equifax-data-breach/index.html.

[2] McLaughlin, J. (2016). New Court Filing Reveals Apple Faces 12 Other Requests to Break Into Locked iPhones. *The Intercept*. Retrieved from https://theintercept.com/2016/02/23/new-court-filing-reveals-apple-faces-12-other-requests-to-break-into-locked-iphones/.

[3] Frenkel, S. (2018). Facebook Bug Changed Privacy Settings of Up to 14 Million Users. *The New York Times*. Retrieved from https://www.nytimes.com/2018/06/07/technology/facebook-privacy-bug.html.