Iris: Third-Party Authentication Service

Akshay Padmanabha
akshayp@mit.edu

Kevin Chen
kyc2915@mit.edu

Surya Bhupatiraju
surya@mit.edu

Thomas Zhang
trzhang@mit.edu

ABSTRACT

Existing centralized identification systems, such as Facebook or Google, offer convenient services for authentication and storing user information that third-party services leverage to reduce friction and streamline the process of creating user profiles and logging in. However, these systems present conflict of interests, and users may be interested in a third-party service that focuses exclusively on authentication. To this end, we present Iris, a web-based, secure solution that resolves the single-point-of-failure concern and provides secure protocols of data transmission and transparency of data usage.

Iris consists of two components: a database that holds users' information encrypted with their passphrases along with a public API to retreive this data, and a client-side browser extension that handles encryption and decryption of the user's information so that the passphrase never leaves the user's local computer. In this paper, we detail how using Iris reduces friction for the website and user, and how a high level of security and authentication is achieved. In addition, we examine the potential threat model and present directions for future work.

Keywords

Iris, Identification, Authentication, Encryption, End-to-End Encryption, Cryptography

1. INTRODUCTION

Centralized identification systems, such as Facebook or Google, offer convenient services for authentication and storing user information. Many web and mobile applications leverage the wide availability of these services to streamline the process of logging in and creating user profiles. However, using these services on third-party services present conflicts of interests, and for many reasons, users may have an aversion to use these services to create user profiles.

Alternatively, users may be interested in using an unbiased, third-party service that is focused exclusively on authentication and providing a centralized identification system. To this end, we propose **Iris**: a web-based, secure, centralized system to allow users to fully control what information they expose to third-party services, and offer full transparency as to what data is collected and how it is used.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
.

2. PREVIOUS WORK

A notable example of work towards an unbiased, third-party identification system is OpenID [6]. OpenID is a bold attempt in this direction, but faced many problems. One of the primary drawbacks include bad communication; it is not clear to end users as to how OpenID works or how it is used. This lack of understanding, along with poor implementation, security vulenerabilities, and trust issues, eventually led to OpenID becoming less and less popular.

Facebook Connect faces similar problems of trust and miscommunication, where people are unsure if the third-party service will post to their timeline, or reveal something personal to an unintended audience.

3. OVERVIEW

Iris consists of two main components. The first component is a central database that stores personal information for each user, encrypted with each user's secret passphrase. A public API allows anyone to retrieve data from the database, but no one except for the user can decrypt his own personal information.

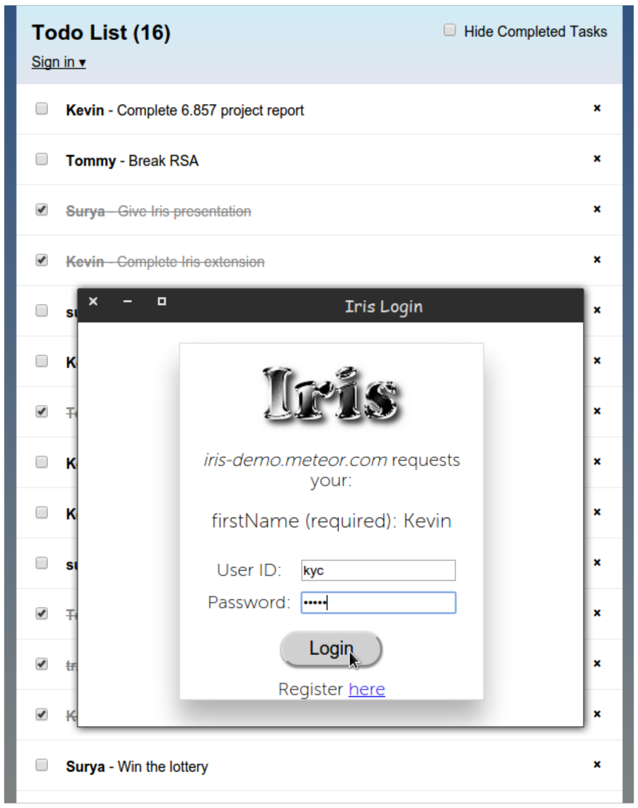
The second component is a client-side browser extension. The extension is responsible for handling encryption and decryption of the user's personal information, so that the user's passphrase never leaves his local machine.

4. GOALS

Iris is designed to meet the following goals.

1. Iris should clearly inform the user about what information a service is requesting, and the user's secret passphrase and personal information not requested by the service should never leave the user's local machine. Consequently, users do not need to trust any third-party service to confidentally store their personal information.

2. The user needs to enter his personal information at most once, after which all his data is stored on the Iris database. This benefits the user as well as the service (users will be more likely to sign up if they do not have to enter a lot of information).



Todo List (16)

Hide Completed Tasks

Sign in ▾

Kevin - Complete 6.857 project report

Tommy - Break RSA

~~Surya - Give Iris presentation~~

~~Kevin - Complete Iris extension~~

Surya - Win the lottery

Iris Login

iris-demo.meteor.com requests your:

firstName (required): Kevin

User ID: kyc

Password: •••••

Login

Register here

Figure 1: A popup appears to 1) indicate that the website is compatible with Iris, 2) present which information the website is requesting of the user, and 3) prompt the user to login or sign up.

3. A service must be able to authenticate a user without itself having to store a database with usernames and passwords. Similarly, the central Iris server must be able to authenticate a user who wishes to edit his information.

5. SECURITY POLICY

Iris is a service both for users to send information to websites safely and for websites to authenticate users securely. Users and websites, however, have no reason to trust each other. The central Iris server also cannot trust any users of its API. In particular,

1. Anyone may use the Iris public API, with arbitrary parameters.

2. Websites may contain arbitrary Javascript code and request any information from the user.

3. Users can modify the code of their Iris browser extension, and the Javascript code on websites.

On the other hand, we assume that users trust the Iris server to store their encrypted data safely; the Iris server has no incentive for malpractice because it cannot decrypt any of its contents. We also assume that information on the user's local machine is safe; for example, we do not consider key-logging attacks or malware installed on the user's machine.

6. USING IRIS

The browser extension is a Chrome extension that must be installed on the user's browser in order to use Iris. Securely sending personal information to a website involves the following steps:

1. The website makes a request to the extension with a Javascript call,

2. The extension creates a popup asking for the user's ID and passphrase,

3. The extension locally decrypts the data retrieved from the Iris database, and

4. The extension sends the requested data and a signature to the website.

We describe each of these steps in detail below.

6.1 Website requests data

The extension injects a Javascript function definition `iris_request()` into every website that the user visits, which the website can call to request user data. The function signature is

```
iris_request(requested_data, callback),
```

where `requested_data` is a dictionary of the form `{required: ['name', 'email'], optional: ['home address']}`. The user must provide all required attributes to use the service, but users may choose to not reveal an optional attribute.

If the user confirms the request, then the callback function will be invoked with a dictionary of the user's personal information, for example `{name: 'Kevin', email: 'kyc@mit.edu', home address: '500 MIT Drive'}`.

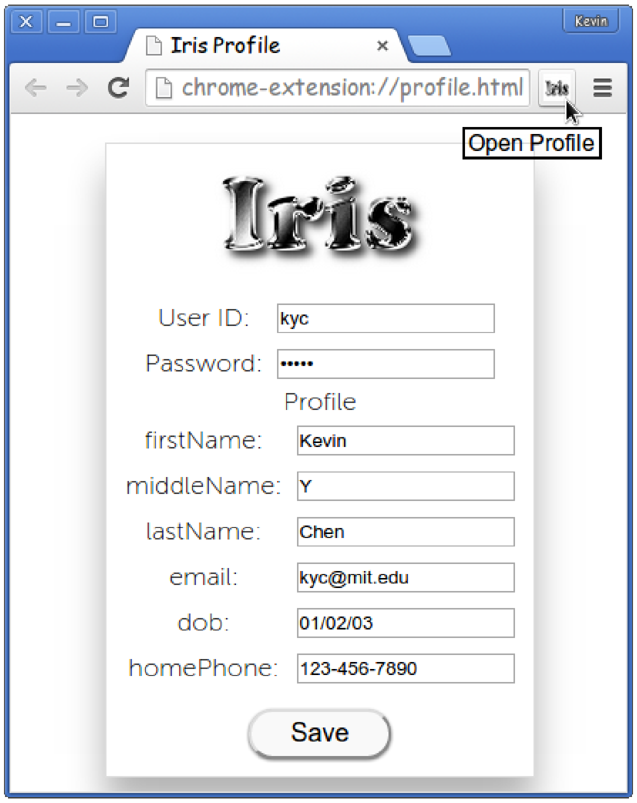
6.2 Popup

When the `iris_request()` function is invoked by the website, the function arguments are passed through a Chrome *message* to the browser extension, which creates a popup window. The popup window is a simple GUI that clearly shows what personal information the website is requesting, and prompts for the UserID and Passphrase. Figure 1 demonstrates the simple GUI and emphasizes the information that the website is requesting.

With the highest security settings, a popup requesting ID and Passphrase is created every time a user visits an Iris-compatible website. However, users can select other settings to cache data. For example, the extension can cache their ID locally. The extension can also cache their personal information. In the latter case, the user does not need to enter his/her passphrase, unless he/she wishes to edit his/her information or customize the information sent to this particular website.

6.3 Data retrieval and decryption

As soon as the user enters his/her ID, the extension makes an asynchronous call to the Iris server to retrieve the encrypted contents associated with this ID. Here is a sample server response:



User ID: kyc
Password: •••••
Profile
firstName: Kevin
middleName: Y
lastName: Chen
email: kyc@mit.edu
dob: 01/02/03
homePhone: 123-456-7890

Figure 2: A user is able to edit his or her information by clicking on the Iris extension icon. The system recognizes the user after he or she types in the password, and is able to edit and save the information.

```
{
    UserID: 'kyc',
    Data: '[encrypted]',
    Encryption params: [json blob],
    Auth: (e, n)
}
```

The `Auth` field is the RSA public key used by websites to verify signatures. The field `Data` is encrypted and decrypted with SJCL (Stanford Javascript Crypto Library), which uses PBKDF2 to convert the user's passphrase into an encryption key and then uses 256-bit AES. The encryption parameters (for example, the IV used for AES) are stored in plaintext in the `Encryption params` field. The decryption is all performed locally by the browser extension.

6.4 Sending requested data to the website

The decrypted data is a dictionary that looks like the following:

```
{
    name: 'Kevin',
    email: 'kyc@mit.edu',
    home address: '500 MIT Drive',
    SSN: '123-456-7890',
    services: {
        [customized data for each website]
    },
    auth key: (p, q, d)
}
```

The data contains various user information. It also contains the `services` field which contains customized data for each website; for example, if a user wanted to use an email for his/her LinkedIn account different from the email for other accounts. Finally, the `auth key` field is the RSA private key used to generate signatures. This is crucial in allowing users to sign in to Iris from different devices - as long as a user has his/her user ID and password, he/she can generate signatures from any device, mobile or otherwise.

The browser extension filters only the fields requested by the website. In particular, the extension always clears out the sensitive `services` and `auth key` fields before sending the data to the website. The extension also adds two additional fields: the user ID, and the RSA signature.

7. AUTHENTICATION

The previous section describes how the user can safely send personal information to a website without revealing his/her private passphrase to the website or any other third-party service. We now describe how the user can authenticate himself/herself to the website, so that the website can trust his/her identity without its own username/password database, as well as how the user can authenticate himself/herself to the Iris server, so that he/she can modify the data stored on the server.

7.1 Authentication to websites

The argument to the callback function in `iris_request` contains a `signature` field, which is a standard RSA signature m^d computed by the extension. The value m is a hash of the message (the user's requested personal information) and the current time, to prevent replay attacks. The value d is the RSA private key, which is known to the extension because it is stored in the `auth key` field of the decrypted data.

The website can authenticate the user with server side code as follows: first, it retrieves the `(e, n)` public key from the Iris server corresponding to the claimed user ID. Then the website confirms the signature by computing $(m^d)^e$.

These two steps are easily packaged into a library function that the website can import on the server side. We wrote a short library function and used it in our demo page to authenticate the user. This allows websites to authenticate users without a username/password database; instead they just run a simple, stateless function.

7.2 Authentication to Iris server

Now suppose the user wishes to edit his/her personal information. The user can do so by clicking on the browser action button of the Iris extension, which reveals a profile page. To edit data, the user modifies the desired field and presses `Save`. Figure 2 illustrates the GUI and shows how users can edit their information.

The Iris server has a `POST` call to edit the data for a given user ID. However, since anyone can make a `POST` request, the Iris server must have a mechanism to only allow a user to edit his/her own entry.

In principle, a signature can be sent just as when authenticating to a website. However, this is susceptible to relay attacks: once a website receives a m^d signature, it can immediately make a `POST` request to the central Iris server before the time changes.

Because of this vulnerability, we use another approach to authenticate `POST` requests: in addition to sending the new data in the body of the `POST` request, the extension must also send the `(p, q)` private key, as well as a new `(e, n)` public key. The server can authenticate the request if and only if $pq = n$, and relay attacks are prevented because the private key changes with every request.

```
POST({old auth key:  (p, q), new auth:  (e, n),
          new data:  [encrypted]})
```

8. THREAT MODEL

There are two main deficiencies in our current system. The first is that no authority exists for initially creating users. The Iris server supports a public `PUT` request to create a user entry, but it can be overloaded. Such an authority is outside the scope of our project, but is necessary if this system is to be put into widespread use in the future. Additionally, our current implementation cannot prevent distributed denial-of-service attacks, but this is again outside the scope of our project.

The second consists of the various ways sensitive information can be revealed. For example, if a user's computer was compromised by a keylogger, our system cannot prevent the user's password from being revealed. Furthermore, if malware changes the extension to record user IDs and passwords, a user's information is again compromised. Finally, if a website poses as a legitimate website, it could trick users and Iris into giving it a user's information.

9. FUTURE WORK

At its current state, the application is ready to be deployed to machines and used by websites to streamline the login process. However, there are multiple directions to increase functionality and availability of the service. Namely, we'd like to create extensions for all popular web browsers to become the de facto method of authentication and to greatly improve the UI of the application, as it was not the focus of this project. Along these lines, we would like to support more clear use cases of authenticating on multiple devices, as this was also outside the scope of our current project. However, both of these features are easily extendable with the current modular design.

In addition, we would like to be able to protect against some of the common attacks as detailed in Section 8, such as implementing features to detect large amounts of network traffic to prevent DDoS attacks, and to store a list of trusted websites to prevent phishing from malicious websites.

10. ACKNOWLEDGMENTS

We would like to thank Professor Ronald Rivest and TA William Cyr for providing helpful advice and directions for our project.

11. REFERENCES

[1] Serge Egelman. 2013. My profile is my password, verify me!: the privacy/convenience tradeoff of facebook connect. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (CHI '13). ACM, New York, NY, USA, 2369-2378. DOI=10.1145/2470654.2481328 http://doi.acm.org/10.1145/2470654.2481328

[2] Thomas Groβ. 2003. Security Analysis of the SAML Single Sign-on Browser/Artifact Profile. In *Proceedings of the 19th Annual Computer Security Applications Conference* (ACSAC '03). IEEE Computer Society, Washington, DC, USA, 298-.

[3] Khan, R.H.; Ylitalo, J.; Ahmed, A.S., "OpenID authentication as a service in OpenStack," Information Assurance and Security (IAS), 2011 7th International Conference on , vol., no., pp.372,377, 5-8 Dec. 2011. DOI=10.1109/ISIAS.2011.6122782

[4] David Recordon and Drummond Reed. 2006. OpenID 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management* (DIM '06). ACM, New York, NY, USA, 11-16. DOI=10.1145/1179529.1179532 http://doi.acm.org/10.1145/1179529.1179532

[5] Blake Ross, Collin Jackson, Nick Miyake, Dan Boneh, and John Mitchell. 2005. Stronger Password Authentication Using Browser Extensions. In *Proceedings of Usenix security* http://crypto.stanford.edu/ dabo/abstracts/pwdhash.html

[6] San-Tsai Sun, Eric Pospisil, Ildar Muslukhov, Nuray Dindar, Kirstie Hawkey, and Konstantin Beznosov. 2011. What makes users refuse web single sign-on?: an empirical investigation of OpenID. In *Proceedings of the Seventh Symposium on Usable Privacy and Security* (SOUPS '11). ACM, New York, NY, USA, , Article 4 , 20 pages. DOI=10.1145/2078827.2078833 http://doi.acm.org/10.1145/2078827.2078833