Abstract

Software engineering is an application of computer science that entails much more than the development of algorithms and the use of data structures. The goal of this project was to delve into software engineering by learning web development. Web development and software engineering is an extremely valuable skill to those in the workforce today, and I wanted to understand the challenges that such developers face today as well as the programming paradigm that these developers use. To accomplish this goal, I set out to build two-factor authentication system and API using Flask and Facebook Messenger bots. Two-factor authentication is a method of access control in which a user is only granted access after successfully presenting several separate pieces of evidence to an authentication mechanism. Flask is a Python web framework that is popular not only because of its simplicity but also because of its scalability, and Facebook Messenger is an online messaging platform that supports over one billion users. Currently, two-factor authentication is most widely used by requiring users to prove their identities with an application on their mobile phones. However, this approach has received criticism due to its slowness and difficult usage. To make the Two-Factor Authentication process more seamless while preserving security, it would be useful to integrate the authentication with an existing secure application with many users. Thus, the use of Messenger bots for two factor authentication provides promise as an easier way to authenticate into online service providers. In this project, I developed an end-to-end secure login system with two-factor authentication in Facebook Messenger. I also developed an API to integrate this authentication as a single-sign on system for other login systems.

Keywords: Two-Factor Authentication, Security, Software Engineering
1. Introduction and Background

Privacy has become increasingly important in an increasingly technological world. Indeed, as more people are going online, we have also seen an increase in the amount of malicious attacks and hacks on unsuspecting users. There have been many ways to combat this. Many companies with login systems have required their users to create passwords that contain a certain number of characters. More recently, the SHA-1 cryptographic hash is becoming deprecated in 2017 in order to migrate to a more secure signing algorithm known as SHA-2, and websites are increasingly using the HTTPS protocol as opposed to the less secure HTTP protocol.

1.1. Two-Factor Authentication

One way that login systems have attempted to increase security is by using two-factor authentication. Two-factor authentication is a method of access control in which a user is only granted access after successfully presenting several separate pieces of evidence to an authentication mechanism. At Yale, we use Two-Factor Authentication with our central authentication service (CAS) login [1]. Users who login on CAS and are not on a Yale WiFi network must verify their login using a mobile app known as DuoMobile. This notion of security is built on the premise that users are intimately tied to their mobile phones, and those who try to log in to CAS can only be authenticated with possession of the phone. However, this is preventative, or at the very least troublesome, for many users because DuoMobile is not an app that comes pre-installed on mobile phones. Therefore, users must download the app and will only use it when logging on, and the app serves no other purpose. In fact, DuoMobile and other Two-Factor Authentication applications have various bugs that negatively affect the user experience, making a process as simple as logging in very complex.

To make the Two-Factor Authentication process more seamless while preserving security, it would be useful to integrate the authentication with an existing secure application with many users. Thus, my project aims to create a Two-Factor Authentication system using Facebook Messenger. We can leverage the incredibly wide user base of Facebook Messenger to make the Two-Factor Authentication much less painful. Indeed, this past summer,
Facebook announced that Messenger has hit over 1 billion users. Thus, Two-Factor Authentication using Messenger would provide much less overhead than having to download an entirely separate application that serves no other functions. On the contrary, integration with Messenger would also help Messenger become a more comprehensive, one-stop-shop application.

To facilitate Two-Factor Authentication in Messenger, we can make use of Facebook bots. In the beginning of the calendar year, Facebook released development support for their Messenger platform by announcing an API for Facebook Messenger Bots. These bots have increasingly become an integral part of the Messenger ecosystem. Companies have been using Messenger bots to handle transactions with users, play games, and follow news, among a myriad of other functions. The Application Programing Interface (API) of these bots contains capability to send and receive text. Facebook also provides support for natural language assistance. Thus, with respect to Two-Factor Authentication, these bots can also play an important role. By allowing bots to message users for authentication, this provides for high scalability. In addition, the use of Messenger bots still provides the second layer of security for Two-Factor Authentication because users are intimately tied to their Facebook and Facebook Messenger accounts, which are securely handled by Facebook.

1.2. Single Sign On

Single sign-on (SSO) is a session and user authentication service that permits a user to use one set of login credentials (e.g., name and password) to access multiple applications. The SSO service authenticates the end user for all the applications the user has been given rights to and eliminates further prompts when the user switches applications during the same session. For instance, Yale CAS is the SSO service for Yale students wishing to access their Google applications (Gmail, Docs, etc.,). In SSO, there exists the notion of service providers and identity providers. A service provider (SP) is an entity that provides Web Services where as an identity provider (IdP) is an online service or website that authenticates users on the Internet by means of security tokens. In the previous example, Google is the SP that is providing the services of Gmail or Google Docs, and Yale CAS is the IdP that is authenticating users to use the services offered by the SP.
Most IdPs and SPs communicate securely via a standard known as Security Assertion Markup Language (SAML). SAML SSO works by transferring the user's identity from the IdP to the SP using digitally signed XML documents. The use of XML documents as opposed to propriety software makes compatibility across different IdPs and SPs seamless and scalable [2]. The IdP and SP exchange SAML protocol messages through the user's browser. The SP sends an SAML authentication request message to the IdP, asking to authenticate the user. The IdP typically asks the user for a username and password (although any other method of authentication can be used) and if the password is correct, the IdP sends back a SAML authentication response stating that the user has just logged in successfully at the IdP, together with some proof that the message was indeed sent by the IdP.

In this project, I also aim to create a secure IdP that can hook up to existing SPs. Many existing SPs such as Gmail and Facebook for Work have a list of supported IdPs such as ADFS (Active Directory Federation Service), Azure AD, G Suite, Okta, and OneLogin.

2. Specification and Deliverables

My goal is to build MessengerAuth, an end-to-end login system that uses Two-Factor Authentication with Facebook Messenger Bots. Users would be able to securely create an account on my platform. Upon login, users would be redirected to a page for two-factor authentication, and they would be prompted to message my Messenger Bot to confirm their identity.

I would also like to be able to configure MessengerAuth as a secure IdP for use not only in my login system, but as an SSO system for any public SP that wishes to provide Two-Factor Authentication through Messenger. Through the process, I also want to learn about web development in Flask, a back-end Python web framework. I also want to learn how to use the Messenger Bot API and parse user input intelligently. Thus, my list of deliverables include the following:

- Secure, end-to-end login system built in Flask
• Server setup and deployment
• Facebook Messenger Bot
• SSO integration with SPs

3. Methods and Implementation

3.1. Web development

The web development programming paradigm involves Models, Views, and Controllers (MVC) [3]. In this programming paradigm, when a client hits an endpoint of a webpage, the endpoint corresponds to a controller of the server. The controller is responsible for translating user input and communicating this input with the models. The models correspondingly represent how data for the web application is stored and abstracted. Once the controller has grabbed the relevant data from the models, this data is presented to the client on the front-end as a view.

In this project, I used the Flask microframework and applied the MVC programming paradigm. Flask uses Python for all back-end logic in its controllers and models, and it uses HTML, CSS, and Javascript to render its views [4]. Flask also makes use of the Jinja2 templating engine to render data sent from the controllers to the view. For the models, I used SQLAlchemy, an open source SQL toolkit and object-relational mapper for Python. In this application, I utilized three models:

• Users - I supported native users within MessengerAuth
• Services - Because SPs could use MessengerAuth for SSO, I kept track of which services are registered with MessengerAuth
• ServiceUsers - One User in MessengerAuth may be tied with multiple Services (e.g. Google, Facebook). Thus, when someone logs into MessengerAuth in SSO to access an SP, that ServiceUser must be registered in the database and connected with a User.

I deployed MessengerAuth using Heroku, a cloud Platform-as-a-Service that supports several programming languages and is used as a web application deployment model. Heroku
uses dynos, lightweight Linux containers that runs a single user-specified command. For my Heroku deployment, I used a free dyno plan, which provides for 550 hours of uptime per month. I also use the Heroku Postgres Database with the Hobby Dev plan, which provides 10,000 rows, continuous protection, direct SQL access, and a connection limit of 20.

3.2. Login Security

One nearly ubiquitous methods that web applications use to make user accounts more secure is by using email confirmation upon user registration. When a user registers for an account using his/her email, a confirmation email is sent such that the user must actually have access to the email in order to successfully create an account. In MessengerAuth, I also implemented email confirmation using SendGrid, a transactional email delivery and management service. The SendGrid Python API enables developers to send templated HTML emails from a single email address [5]. When a user receives the confirmation email, there is a button that will redirect to a URL in MessengerAuth that successfully confirms the user's account. To do this, I generate a cryptographically secure and unique token for the user and pass this token into the URL for the button in the email. The user's account is confirmed if and only if the user visits the confirmation URL that contains the token that is unique to the user. This security system prevents malicious third-parties from arbitrarily confirm user's accounts.

During user registration, I also make use of password hashing in the database. To do this, I use Python's uuid module to hash a user's password. A UUID (Universally Unique IDentifier) is 128 bits long, and can guarantee uniqueness across space and time. I also allow users to change their passwords. To confirm a password change, I once again use SendGrid to send the user an email making sure that he/she indeed requested to change passwords.

3.3. Facebook Messenger Bots

Facebook provides an API to develop automated bots using the Messenger platform. This API enables developers to create bots that send and receive messages to various users that message the bot. Development using the Messenger API requires creating a Facebook page that hosts the bot and is tied to the bot. It also requires setting up a webhook callback
URL for verification. To properly set up the webhook, Facebook sends a GET request to this URL with information in JSON format. The webhook should correspondingly look for the Verify Token within the JSON and respond with the challenge sent in the verification request. Once Facebook subscribes the webhook to the bot, we can start listening for POST calls at the webhook. In other words, all messages sent to the bot will send a callback to this webhook. The payload contains information that includes the Facebook ID of the sender and the message sent. On the back-end, we can process this payload and correspondingly send an acknowledgements and messages back to the user.

To setup Two-Factor Authentication, all users logging into MessengerAuth must have a corresponding Facebook account that is tied to the MessengerAuth account [6]. In particular, every user on Facebook has a unique Facebook ID (FBID), but this information is opaque to the user, so it is infeasible to ask users for their FBIDs upon signup. To successfully link a MessengerAuth account with a Facebook account, I generate a unique, random hash for each user. Upon logging in for the first time with their password for MessengerAuth, the user is given his/her random hash and is directed to message the random hash to the Messenger Bot. Once again, the payload that is sent to the webhook contains the user's FBID. I then tie the FBID with the corresponding user, and a success message is returned (Figure 1a). If a false hash was sent to the Messenger bot, the bot will respond with a message that no account exists with that hash (Figure 1b).

Once the MessengerAuth account has been tied with a FBID, users can begin logging in and taking advantage of Two-Factor Authentication. Upon subsequent logins, users are asked to message the bot with any text. Once the webhook receives any payload from the user's Facebook account, the user is automatically logged in to MessengerAuth (Figure 1c). To increase security, I only allow authentication messages to come within a five-minute period of logging in to MessengerAuth. Indeed, if the user can message the bot for any infinite amount of time in order to pass through two-factor authentication, then if the user's Facebook account is ever compromised, malicious third parties can log in to MessengerAuth at any time by messaging the bot. Thus, in my implementation of the time limit, if messages are received past the five-minute period, the Messenger Bot responds with a message that the
user has failed to Two-Factor authenticate in time (Figure 1d). I do this using a timestamp that is generated for the user on his/her last login, and upon receiving messages in the payload, I compare the current timestamp to the last login timestamp for the user to ensure that the message came within a five-minute period.

Figure 1: (a) Successful association of Facebook account with MessengerAuth account, (b) Failed association of Facebook account with MessengerAuth account due to incorrect hash, (c) Successful Two-Factor Authentication, (d) Failed Two-Factor Authentication within 5-minute time limit

3.4. Single Sign On

With the Two-Factor Authentication infrastructure in place, we can now begin implementing MessengerAuth as an IdP. As mentioned in Section 2, the job of the IdP is to confirm the identity of a particular user so that he/she can use the service provided by SP [1]. The
user starts by providing an email address at the SP, is then redirected to the IdP, and upon authentication, is finally redirected back to some URL of the SP. This final redirect URL is known as the Assertion Consumer Service (ACS) URL. The ACS URL must be specified for each SP.

In order to set up MessengerAuth as an IdP, the SP must also have a private RSA key and provide the corresponding public key to MessengerAuth. Upon authentication in MessengerAuth, the email of the authenticated user is passed back to the ACS URL of the SP. To prevent any malicious third party from hitting the ACS URL with the plaintext email of the authenticated user, it is necessary to securely hash the email of the authenticated user with the provided RSA public key of the SP. Otherwise, any third-party would be able to hit the ACS URL pretending to be the IdP and successfully login to the SP as any particular user. Upon receiving the hashed email, the SP can then decrypt the hash using its private key and then begin allowing the user to interact with the service.

SPs are able to hook up with MessengerAuth as an IdP by sending a POST request to the “/configure_sso” endpoint. The SP must include such information as the name of the SP, the ACS URL, and its public RSA key. I then create a new row for the SP in the Service database table. The SP must then configure its login to redirect to MessengerAuth with the following URL:

```
.cs490-project.herokuapp.com/login?service=<service>&service_email=<email>
```

The user of the SP is then re-directed to a login page for MessengerAuth. The user of the SP must also have a corresponding account on MessengerAuth. Once the user logs in for the first time using the SP, I create a corresponding ServiceUser that ties the user of the SP with the particular account on MessengerAuth. From this point forward, the user can complete the login flow as specified in previous sections. After successful authentication, the user is re-directed back to the ACS URL, and it is the job of the SP to verify the hash and successfully enable the user to start using the service.

To test SSO with an SP, I worked with YHack, Yale’s premier hackathon, to integrate MessengerAuth as an IdP. The YHack website supports thousands of applicants per year, so
the integration of an IdP that provides Two-Factor Authentication has far-ranging implications for security. In configuring YHack as a SP for MessengerAuth, I worked with developers from YHack in integrating an endpoint that sends a request to the MessengerAuth login and in setting up the ACS URL for successful authentication back into YHack.

4. Discussion

4.1. Challenges

There were a number of challenges that I encountered in the development of MessengerAuth. To begin, it was difficult to get started with Flask because the MVC programming paradigm is very different from what traditional computer scientists are accustomed to. Figuring how to set up and hook the models, views, and controllers with each other required looking at many examples and going through many tutorials. There were also various roadblocks related to web programming that arose. For instance, the concept of user sessions is an important part of web programming in all languages (NodeJS, Ruby on Rails, etc.,). I had difficulty in Flask figuring out how to handle user sessions for logging in and logging out. Also, I spent a lot of effort in creating a seamless user interface that was responsive on not only desktop but also tablets and mobile devices. The art of making designs responsiveness is increasingly important as the number of devices increases.

Another challenge was getting set up with the Messenger API. As stated in Section 3.3, linking a Messenger Bot with a web application requires the creation of an endpoint that Facebook sends a verification token to. However, Facebook requires that the URL for the webhook is an https endpoint. However, running the web application on localhost is merely http by default, so in order to proceed with the Messenger Bot, I had to deploy the web application to Heroku and set up SSL for https support.

Finally, setting up MessengerAuth as an IdP was particularly challenging. The original goal was to setup MessengerAuth as an IdP to Gmail. The motivation behind this decision was that Yale Central Authentication Service and DuoPush Two-Factor Authentication is configured with Gmail. Setting up an external IdP with Google involves create a G Suite business and routing all emails that end with a certain handle to a particular URL. For
instance, Yale has its own G Suite account, and all @yale.edu email logins re-direct to the Yale CAS IdP login. In an attempt to setup MessengerAuth as an IdP, I created my own G Suite business and initially routed all emails to the login page for MessengerAuth. However, as stated in Section 2.2, Google uses the SAML protocol to securely pass the identity of the user to and from the SP and the IdP. Working with SAML, however, would require extensive integration with deployed federated identity solutions such as Shibboleth. Google also has a set of trusted IdPs that are easily configurable with Google SSO, but developing one’s own IdP involves extensive overhead.

4.2. Future Work

One avenue of future work involves formally submitting this Messenger Bot to Facebook for approval. In this way, MessengerAuth will be publicly available to all users who wish to use Messenger for Two-Factor Authentication. Currently, because the Messenger Bot has not been formally approved, only my personal account is able to message it and receive responses.

In addition, when a user is on the page that requests two-factor authentication using Messenger, the client continually polls the database for whether the user has indeed authenticated on Messenger. In this way, on a database change when the user authenticates, he/she will be automatically re-directed to the success page. However, this approach is not very scalable if there are hundreds of users polling the database at the same time. Instead, an optimization that could be made is if the back-end, upon changing the database, sends a response to the client. Upon receiving the response, the client can then automatically trigger the re-direct. This is much more efficient because it scales infinitely with the number of clients [7].

Another improvement that could be made is the integration of SAML using Shibboleth so that MessengerAuth can be used by SPs such as Gmail, Facebook at Work, and the myriad of other SPs that require SAML for SSO. This would, for instance, enable Yale students and faculty to use MessengerAuth for Two-Factor Authentication instead of using DuoPush.

Finally, MessengerAuth currently exists as a web application that is responsible on mobile devices, but perhaps it would also be helpful to develop a mobile application in iOS or
Android for MessengerAuth. This would be analogous to Duo Mobile, a premier Two-Factor Authentication mobile application that is currently integrated with the Yale CAS login. Developing a mobile application would also be relevant to today’s market as more and more people around the world get smart phones and we see the shift from web-first to mobile-first designs.

5. Acknowledgments

A special thank you to Professor Ruzica Piskac of the Yale Computer Science Department for advising this project. She has given me suggestions, feedback, and direction through the entirety of this endeavor.

6. References


