Production Requirements Document

Team: Not Our (Seg)Fault
Project: AuthentiKey

1/12/17

Contents

1 Introduction

Team Members: Alex Wein (lead), Ashlynn Cardoso (scribe), Clara Frausto, Caitlin Scarberry, William Bennett

Team Mentors: Alec Harrell, Renato Untalan

1.1 Problem & Background

Computer systems are inseparable from modern daily life, making secure authentication more vital than ever. However, despite its many flaws, the most common authentication method remains the password. Users frequently reuse passwords across different platforms, which increases the severity of database breaches; users often set insecurely short passwords to make them easier to remember; and once a password has been acquired by a malicious party, that party can immediately use the password. Additional authentication is clearly required for a secure system.

1.2 Innovation & Objective

The end goal of our project is a robust authentication system using keystroke dynamics. Keystroke dynamics, which analyze how a user types, are a biometric authentication method that avoids the pitfalls of both methods listed above. The vast majority of users interface with their devices through a keyboard, whether that is an on-screen mobile keyboard or the physical keyboard of a personal computer. This eliminates the need for separate devices or specialized hardware.

1.3 Assumptions

- The majority of users have both access to and the ability to use a physical keyboard. This is obviously not true for every user, and so Authentikey will not be able to replace standard two-factor authentication for users who, for example, primarily use mobile devices or use text-to-speech as their main form of input.
- A user's typing style uniquely identifies them.
- A user's typing style will not dramatically change, so Authentikey will need to be retrained if a user, for example, sustains a serious arm or hand injury.
- Some portion of users will find typing a randomly generated sentence less cumbersome than using a separate device for two-

factor authentication, and so would prefer our system over most current methods of 2FA.

2 Implementation

Our high-level architecture is diagrammed below, and is structured as follows: we expose a front facing API that will integrate with an end service and capture the necessary data. The data gets relayed to our back-end server, which will communicate with our machine learning cluster to construct models of keyboard metadata and use those models to verify a user's identity. Our back-end server also communicates with a DB that stores user data, such as a unique user ID and the verification challenges that have been issued for that user.

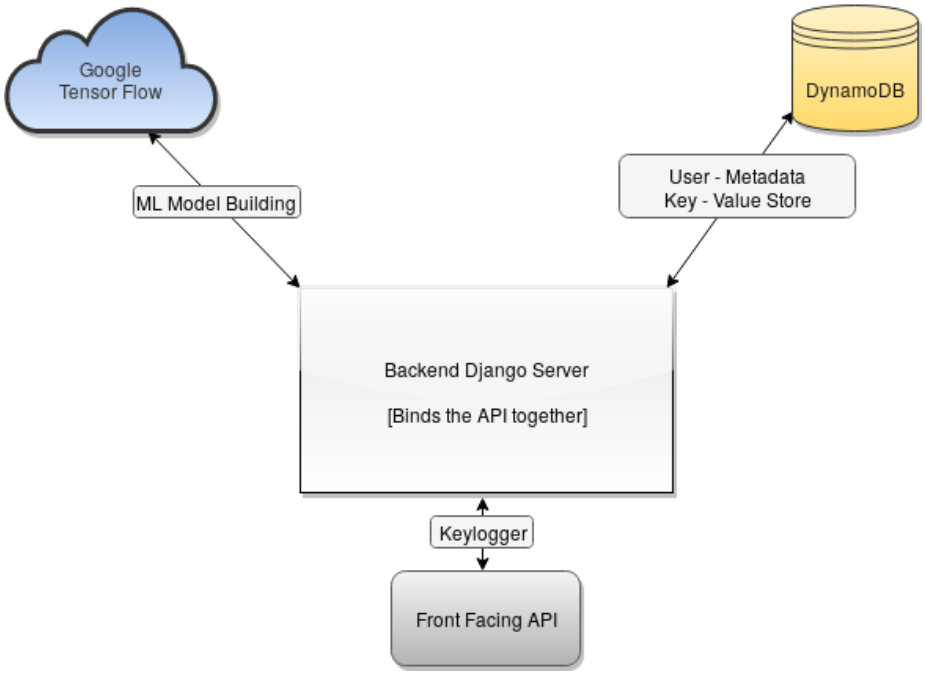

Figure 1: High Level Architecture Diagram

2.1 Metadata Logging

The first component of our system is our keylogger, which records the dwell time (how long a key is pressed) and the flight time (time

between keypresses) while a user types a randomly generated challenge phrase. This is the front facing API of our system. It will be able to be integrated in various login flows, for now we have integrated it with our own front end web portal.

2.2 Machine Learning Cluster

The machine learning cluster, via Google Tensorflow, does the bulk of the data processing where it builds models from user's training data and performs classification based on existing models. The cluster is treated mostly as a black box from the perspective of the rest of the system, as it just returns a probability of whether or not the user who typed the string is who they claim to be.

2.3 Data Storage

We use a DynamoDB living on AWS for storage of all our user data. NoSQL is used due to a lack of schema: we store userIDs as keys and a binary blob representing their corresponding ML model based on their keyboard metadata.

2.4 Back-end Server/Internal API

The backbone of our system is a Django server that acts as an interface between all of the different services/systems we use. Firstly, it processes the raw metadata input from our front end API and parses and serializes it before serving it up to our ML cluster. It also manages our data storage by making requests to add, remove, update, and verify users. In this case both the ML cluster and our DB have internal API wrappers which our server uses for communication and management.

2.5 System Models

Class Diagram

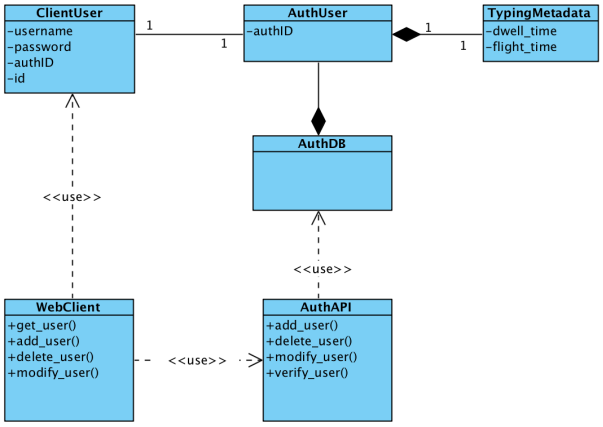

Figure 2: Class Diagram

Sequence Diagram For Logging In:

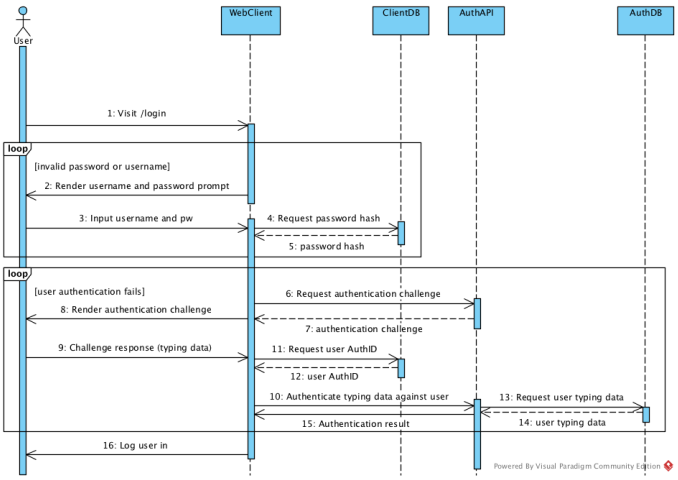

Figure 3: Sequency Diagram: Logging in

Sequence Diagram For Activating or Updating AuthentiKey 2-Factor Authentication:

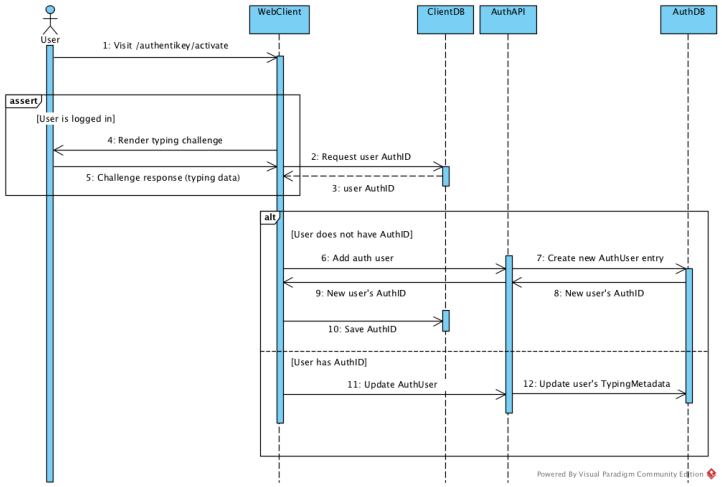

Figure 4: Sequence Diagram: Activating or Updating AuthentiKey 2-Factor Authentication

3 User Interface

When a user attempts to log in with AuthentiKey, after entering a valid username and password, they will be presented with a short typing challenge:

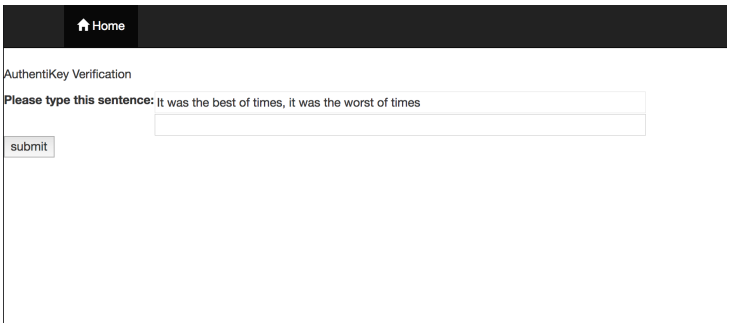

Figure 5: Login using Authentikey

When a user attempts to activate AuthentiKey for their account, they will be presented with a longer prompt:

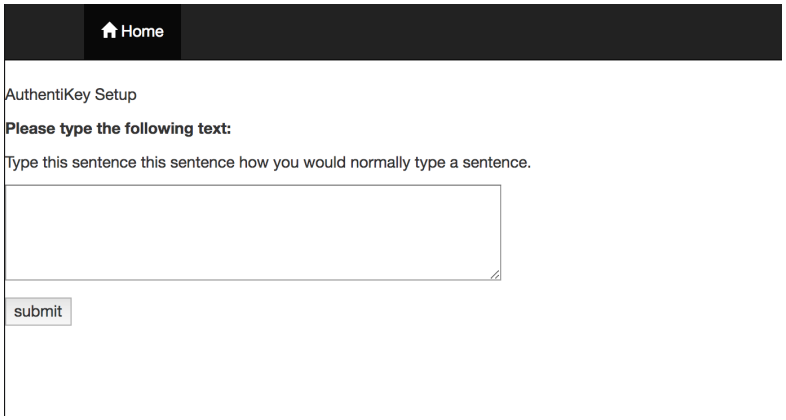

Figure 6: Activation of Authentikey

4 Requirements

4.1 User Stories

1. As a user, I want to log in using AuthentiKey as a second factor of authentication.
   (a) Acceptance Test: Upon entering correct username and password at a login portal, the user is presented with a typing challenge which must be passed to log in.
   (b) Implementation: https://github.com/caitscarberry/django-authentikey/commit/927708da62bb4e37ae2f52a3168bc934b2290515
2. As a developer, I want to be able to add a new user and add their metadata to our backend.
   (a) Acceptance Test: Upon return of a model from the ML cluster the user should be added into the user table on the database with a new uid that is returned from the API
   (b) Implementation: https://github.com/MadRubicant/not-our-segfault/commit/8f8e2ac7f5d1f058f36d53aa5bad32405c3bc007

3. As a developer, I want to be able to delete a user's username and metadata from our backend.
    (a) Acceptance Test: Upon making a request to delete a user, for whatever reason, the API, provided a uid, should remove the user from the database and return a success code (True/False)
    (b) Implementation: https://github.com/MadRubicant/not-our-segfault/commit/8f8e2ac7f5d1f058f36d53aa5bad32405c3bc007
4. As a developer, I want to be able to update a user's metadata so that the user can retrain if their typing style changes.
    (a) Acceptance Test: Upon retraining a user and getting a response from the ML cluster the user, corresponding to uid, should have their metadata updated in the database and the API should return a success code (True/False).
    (b) Implementation: https://github.com/MadRubicant/not-our-segfault/commit/8f8e2ac7f5d1f058f36d53aa5bad32405c3bc007
5. As a developer, I want to be able to validate a user based on their keyup and keydown time.
    (a) Acceptance Test: Upon receiving user input from the challenge phrase the database should be able to compare data returned from the ML model to the metadata stored in the database and return a decision regarding the validity of a user (True/False).
    (b) Implementation: https://github.com/MadRubicant/not-our-segfault/commit/8f8e2ac7f5d1f058f36d53aa5bad32405c3bc007
6. As a user I want to see a new, normal looking, challenge sentence phrase every time I authenticate.
    (a) Acceptance Test: A user logs in once using AuthentiKey, logs out, and logs in again. The challenge phrase presented in each login should be unique.
    (b) Implementation: To do, with high priority.

7. As a user I want to have a low rate of failed authentications due to the failure of the machine learning/system.
    (a) Acceptance Test: When a user tries to log into their own account, nine out of ten authentication attempts should allow them access.
    (b) Implementation: To do, with medium priority.
8. As a developer/admin I want to minimize the times a user is falsely authenticated to maximize security.
    (a) Acceptance Test: When a user attempts to log into an account that is not theirs, nine out of ten authentication attempts should fail.
    (b) Implementation: To do, with medium priority.
9. As a developer I want the machine learning model to be able to utilize metadata on all possible keys on the keyboard (viz. Keyboard agnostic).
    (a) Acceptance Test: A user using a non-qwerty keyboard is able to activate AuthentiKey and use it as a second factor in logging in, with an 80% success rate.
    (b) Implementation: To do, with high priority.
10. As a user, I want to be able to activate AuthentiKey as a form of two-factor authentication.
    (a) A user is able to go to a page where they are presented with a typing challenge. After completing and submitting the challenge, the user should be able to log in using AuthentiKey as a second factor.
    (b) Implementation: To do, with high priority.

11. As a user, I want to be able to activate AuthentiKey as a form of two-factor authentication.
    (a) Acceptance test: A user is able to go to a page where they are presented with a typing challenge. After completing and submitting the challenge, the user should be able to log in using AuthentiKey as a second factor.
    (b) Implementation: https://github.com/caitscarberry/django-authentikey/commit/ba7a67ef62ead4f5b28044bfb78d07e3bb41391a
12. As a developer, I want a user to be presented with an AuthentiKey challenge only if that user has activated AuthentiKey.
    (a) Acceptance test: A user who has not activated AuthentiKey can log in with only their username and password. A user who has activated AuthentiKey will be presented with a typing challenge.
    (b) Implementation: https://github.com/caitscarberry/django-authentikey/commit/10cb0efd3b85f1255635f0e6311fa114417cc7da
13. As a developer, I want to be able to use ML to remove outliers from metadata that a user passes.
    (a) Acceptance test: After the ML combs through the metadata, there should be no extreme outliers in the user's metadata.
    (b) Implementation: https://github.com/MadRubicant/not-our-segfault/commit/535d051e17c0799769ebe576e3869e165bb7460
14. As a developer, I want each user to have an unique threshold value that accounts for their individual consistency.
    (a) Acceptance test: Each user's threshold value should be calculated based on how much variance a user has with multiple authentications, rather than some hard coded value.
    (b) Implementation: To do, with medium priority

15. As a developer, I want the ML to be able to update the threshold value with each successful authentication.
    (a) Acceptance test: A user should be able to consistently authenticate themselves despite increasing variance in keystrokes over time.
    (b) Implementation: To do, with medium priority
16. As a user, when I log in, I want to know before completing the typing challenge if AuthentiKey is down.
    (a) Acceptance test: While the AthentiKey server is down, a user who has AuthentiKey activated and gets past the username/password prompt should be presented with an error message.
    (b) Implementation: https://github.com/caitscarberry/django-authentikey/commit/10cb0efd3b85f1255635f0e6311fa114417cc7da
17. As a developer, I want ML to create a new challenge string from a corpus of my choosing.
    (a) Acceptance test: The ML can use words/phrases from the chosen corpus to generate a challenge string.
    (b) Implementation: https://github.com/MadRubicant/not-our-segfault/commit/3322a5a42937c7119ef31ff7e66ebdafb7eeaa28
18. As a developer, I want to be able to split one large file of a user's metadata to track changes within one large paragraph.
    (a) Acceptance test: Given a five and a denomination to split, the function returns that number of pieces of metadata.
    (b) Implementation: https://github.com/MadRubicant/not-our-segfault/commit/b6eb1feb557047e18b129e3a58c63228fae82868
19. As a developer, I want to use a fuzzy extractor to securely store metadata on the server.
    (a) Acceptance test: The fuzzy extractor successfully maps the metadata into a format that is deterministically hashable.
    (b) Implementation: To do, with low priority.

20. As a user, I want to be able to disable AuthentiKey as a form of two-factor authentication.
    (a) Acceptance test: By visiting /authentikey/deactivate, a logged-in user is able to deactivate AuthentiKey. After this, they will be able to log in without being presented with a typing challenge.
    (b) Implementation: To do, with low priority.

Appendix

A Technologies Employed

- Django
- AWS
- TensorFlow
- DynamoDB
- JavaScript/HTML/CSS
- Python