BROWNFOX PROJECT

TYLER SENTER, BRYCETON BIBLE, KINCAID MCGEE

REAL-TIME USER AUTHENTICATION

VIA UNINTRUSIVE TYPING METRICS

Collect & Build

User model in realtime

Verify Identity

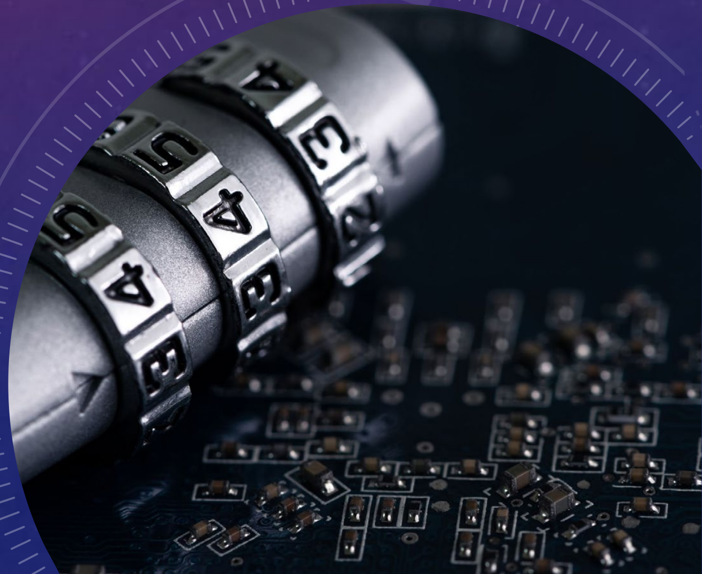
And alert to potential security breaches

Using JavaScript

Lightweight & embeddable

Machine Learning

Opens future potential



OBJECTIVE: TOOL FEATURES

- Collect & maintain a model via natural typing input from a user
- Classify an unknown user to secure a webpage
  - If a user is unknown, request further authentication – if success, update model.
  - If a user is known, work quietly in the background
- Run in the background to allow user to naturally input text

OBJECTIVE: IMPLEMENTATION GOALS

- Use a lightweight, portable web language for all tool features
- Collect information that is non-platform dependent
- Classify user in a way allowing for future model training-feature expansion

IMPLEMENTATION DETAILS

ML.JS

Machine learning library

Node.js & JavaScript

For keystroke detection & data processing

K-Nearest Neighbors

For classification & prediction

IMPLEMENTATION DETAILS: 3 STAGES

1. Collect all keystroke events
2. Process events and generate features
3. Train or classify on generated features

STEP 1: KEYSTROKE EVENT COLLECTION

Using JavaScript *keydown* and *keyup* events, collect information about each keystroke:

- Keycode
- Modifier keys
- Repetitions
- Timestamps
  - Start/Stop
  - Duration

```
Console was cleared                                                                                         session.js:60
Key Events                                                                                                  session.js:61
▼Object ℹ                                                                                                   session.js:63
    key: "h"
    keyCode: "KeyH"
  ▶keypress: {alt: false, ctrl: false, meta: false, shift: false, repetitions: 0}
  ▶timing: {start: 1638782199739, stop: 1638782199826, duration: 87}
  ▶[[Prototype]]: Object
▶Object                                                                                                     session.js:63
▶Object                                                                                                     session.js:63
▶Object                                                                                                     session.js:63
▶Object                                                                                                     session.js:63
▶Object                                                                                                     session.js:63
▶Object                                                                                                     session.js:63
▶Object                                                                                                     session.js:63
▶Object                                                                                                     session.js:63
▶Object                                                                                                     session.js:63
▶Object                                                                                                     session.js:63
Word Events                                                                                                 session.js:65
                                                                                                            session.js:67
▼Object ℹ
  ▶eventTimings: {wordStart: 1638782199739, wordEnd: 1638782200394, duration: 655, downtimes: Array(4), totalDowntime: 248}
    finalString: "hello"
  ▶keySequence: (5) ['h', 'e', 'l', 'l', 'o']
  ▶keyTimings: {KeyH: {…}, KeyE: {…}, KeyL: {…}, KeyO: {…}}
    numMistakes: 0
  ▶[[Prototype]]: Object
▶Object                                                                                                     session.js:67
```

STEP 1: KEYSTROKE EVENT COLLECTION

Using JavaScript *keydown* and *keyup* events, collect information about each word:

- Timestamps
    - Start/end
    - Duration
    - “Downtime” (time spent not pressing a key during a word)
- Estimated mistakes
- Estimated final key sequence

```
Console was cleared                                                                              session.js:60
Key Events                                                                                       session.js:61
▼Object                                                                                          session.js:63
    key: "h"
    keyCode: "KeyH"
  ▶keypress: {alt: false, ctrl: false, meta: false, shift: false, repetitions: 0}
  ▶timing: {start: 1638782199739, stop: 1638782199826, duration: 87}
  ▶[[Prototype]]: Object
▶Object                                                                                          session.js:63
▶Object                                                                                          session.js:63
▶Object                                                                                          session.js:63
▶Object                                                                                          session.js:63
▶Object                                                                                          session.js:63
▶Object                                                                                          session.js:63
▶Object                                                                                          session.js:63
▶Object                                                                                          session.js:63
▶Object                                                                                          session.js:63
▶Object                                                                                          session.js:63
Word Events                                                                                      session.js:65
                                                                                                 session.js:67
▼Object
  ▶eventTimings: {wordStart: 1638782199739, wordEnd: 1638782200394, duration: 655, downtimes: Array(4), totalDowntime: 248}
    finalString: "hello"
  ▶keySequence: (5) ['h', 'e', 'l', 'l', 'o']
  ▶keyTimings: {KeyH: {…}, KeyE: {…}, KeyL: {…}, KeyO: {…}}
    numMistakes: 0
  ▶[[Prototype]]: Object
▶Object                                                                                          session.js:67
```

STEP 2: DATA PROCESSING

wpm	akl_A	akl_B	akl_C	akl_D
49	114	158	89	66
103	154	135	130	100
57	134	156	102	130
111	92	61	115	72
140	114	73	150	119
111	145	61	63	119
137	139	141	71	141
52	113	134	152	103

Using collected keystroke events, model features are generated.

- WPM
- Average keypress time (overall/individually)
- Downtime

STEP 3: MODEL TRAINING & CLASSIFYING

Training

Store features,
labeled to current,
known user

Classifying

Predict based on
Current collected features

STEP 3: MODEL TRAINING & CLASSIFYING

Classifying is the "authentication step" – it shows what the tool thinks the user's identity is. It should be run when we want to verify a user's identity. We classify using KNN via ML.js.

- Upon login as 2FA
- Upon period of inactivity
- Periodically, especially if suspicious activity is detected
- Constantly

INITIAL TESTING

To test feasibility of this defense, we first generated a dataset.

- Using Sheets, we generated a table of demonstration “models”, with random feasible values
- A model was then trained on this table.

B2 fx =RANDBETWEEN(60,160)

	A	B	C	D
1	wpm	akl_A	akl_B	akl_C
2	49	114	158	89
3	103	154	135	130
4	57	134	156	102

INITIAL TESTING

Then, we tested the classification results

- First, the features for “Person 53” were tested
  - Predicted accurately
    ```
    Classification result:
    [ 'Person 53' ]
    ```
- Then, a new fake person (generated with the same Sheets formulas)
  - Predicted accurately
    ```
    Classification result:
    [ 'Person 6' ]
    ```
- Then, “Person 53” with some features manually slightly changed was tested
  - Predicted accurately
    ```
    Classification result:
    [ 'Person 53' ]
    ```
- Last, “Person 53” with random noise applied to all features
  - Predicted accurately
    ```
    Classification result:
    [ 'Person 53' ]
    ```

```
//Person 53
test = [[82,71,124,72,132,72,75,66,147,125,118,103,
//Random Person
test_fake = [[110,76,127,101,129,104,117,152,77,148
//Slightly shifted from Person 53
test_shift = [[92,71,104,72,132,72,75,66,147,125,11
//Random shifted from Person 53
test_random = test;

min=0;
max=25;
```

USER TESTING

For user testing, we generate models on the code running on a server via VSCode

- Users are prompted with a sentence from one of 8 sets
  - Pangrams (the famous fox sentence)
    - Original case (The quick brown fox)
    - Uppercase (THE QUICK BROWN FOX)
    - Lowercase (the quick brown fox)
    - Titlecase (The Quick Brown Fox)
  - Random sentences (may not contain all letters)
    - Same four cases as above
- A model is created and stored based on the sentence
- Models are aggregated into a matrix for later use in classification
- When a model is available, classification is available

DEMONSTRATION

Let’s test classification!

RESULTS

- Initial viability test with random values successfully predicted even on noisy test cases
- Real-world tests showed success in authentication & differentiation between users via classification
- With tweaking of hyperparameters & sensitivity, tool is ready to implement in real world scenarios and expand to other platforms

RESULTS

Example of model data as stored in CSV

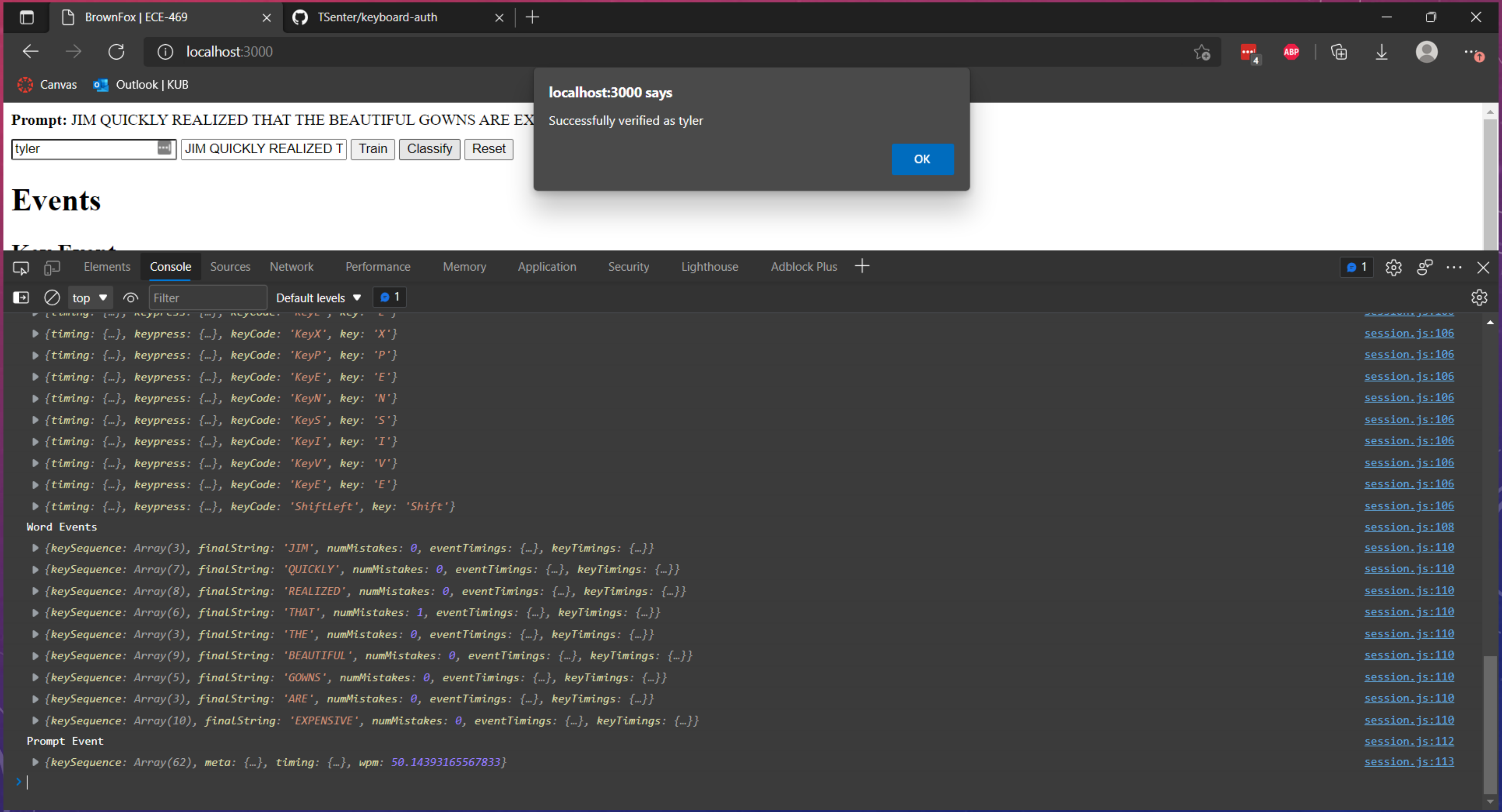
model.csv M   JS session.js M

src > data > model.csv

```
145.5,140,0,142.66666666666666,128.25,105.8,142.5,0,76,0,140,0,1,-86.25,tyler
,120,151,158,132,139,118,134,117.5,4,-915,tyler
33,148.33333333333334,157,116,159,176,106.5,157.5,111,113.66666666666667,124,93,115,4,-136.4,tyler
0,150.25,0,165.2,232.66666666666666,0,0,206.66666666666666,191.5,154.6,0,0,188.33333333333334,0,171,0,2,-1279.2,kincaid
46,205,173,174,238,153.5,185,181,168,4,163.33333333333334,kincaid
33333334,0,0,189,0,209.33333333333334,177,147.33333333333334,0,170,213,143.5,0,0,0,154,128.33333333333334,0,5,-1352.2222222222222,kincaid
5,117,125,104.25,112,127,123.5,119,146,11,393.42857142857144,bryce
33333333334,113,0,131,137,125.75,129,0,146,0,0,0,6,75.54545454545455,bryce
132.25,131,114,128.5,131,114.5,70,135,125,137,99,131,6,-117.45454545454545,bryce
14,79,184,180,99,101,101,0,-196.28571428571428,bryce
,0,118,114,114,0,0,118,0,98,0,0,-91.5,bryce
.5,126.5,93,120,143,130,127,122,121,0,-2.5714285714285716,bryce
41,150,117,111.5,199.5,158,108,143,123,163,0,4.545454545454546,tyler
```

RESULTS

Example of successful auth



Where can this technique go in the future?

- Collecting more features
  - Mouse movements
  - Gyroscope/accelerometer data
  - User agent info
  - More keys, differentiating between capital letters
    - “Red flag keys” – like ￥ for US users and $ for JP users
- Smart analysis of features
  - Analyzing text content with ML-based text comprehension
  - Detecting unusual inputs
    - “Red flag words”
- Multi-platform implementations
  - PCs
  - Smartphones
  - Tablets
  - VR Devices
  - Wearables

LOOKING FORWARD

APPLICATIONS OF THIS TOOL

- 2FA upon login
  - After login via password, require a match
- Authentication before saving secure documents, sending emails, publishing articles
  - Require a match before allowing sensitive tasks to be carried out
  - If match fails, require a secure pin/password (then retrain)
- Periodically verifying identity on chat apps and social media
  - Raise a red flag if a user is typing uncharacteristically
- Demographic targeting
  - Predict user demographics based on their typing metrics, potentially useful to advertisers
- Constant identity verification
  - Raise a red flag on a high-security system immediately when necessary
- Task identification
  - Ensure that monitored user is fulfilling a certain task (working, programming, chatting, typing in a given language)

POTENTIAL SHORTCOMINGS/VULNERABILITIES

- Model must be trained to accommodate all of user's use cases
  - Chatting on SNS vs writing business email
- Tool can only handle alphanumeric keyboards, i.e., standard QWERTY/Dvorak/AZERTY
  - Could be theoretically applied to other languages, but non-romaji hiragana/non-pinyin Chinese input untested
- A good balance of model sensitivity is required
  - Hyperparameters have not been fine-tuned (KNN's K, when to raise red flags)
  - Potentially intrusive
- Some input factors could lead to model being inaccurate – this is a potential security threat
  - Speech input, handwriting input
  - Editing written text after moving cursor with mouse or arrow keys
  - Fixable, but difficult
- Other classification techniques may be more useful

THANK YOU! QUESTIONS?

CONTRIBUTIONS:

TYLER SENTER: NODE BACKEND & UI,

BRYCETON BIBLE: FEATURE SELECTION & ML CLASSIFICATION

KINCAID MCGEE: TRAINING & PROJECT MANAGEMENT

BBIBLE3@VOLS.UTK.EDU

TSENTER@VOLS.UTK.EDU

KMCGEE11@VOLS.UTK.EDU