Adversarial Touch Dynamics

Sean Curran securran@stanford.edu

1 Introduction

Researchers have been experimenting with methods for authentication based on user dynamics, such as keystroke dynamics, mouse dynamics, and now touch-gestures, for more than 25 years, see [3]. Obviously a system like this could be vulnerable to replay attacks and some researchers have even provided solutions to help prevent replay attacks, such as in [1]. It is quite clear that if an adversary acquired raw user input, then the adversary could easily replay the user's raw input and authenticate as that user. However few researchers have considered generating raw user input from the features stored in the database. In fact, in [1] they consider storing the features in a database, as opposed to raw strokes, to be a form of security.

In this article, I demonstrate how a user's feature vectors can be used to counterfeit raw strokes for that user. I use a kNN, with a few tricks, to attack both a kNN classifier and an SVM classifier. The attack requires the adversary to collect touch gesture data, but this is hardly an impedement with the abundance of mobile applications that exists today. Moreover, machine learning algorithms depend on a notion of closesness, thus most machine learning classifiers are susceptible to this type of attack. For this reason, machine learning based authentication mechanisms need something that is analogous to the hashing of passwords.

2 Dataset and Features

2.1 The Dataset

The dataset used contains almost 1 million samples of raw sensor data, which correspond to over 20,000 touch gestures, from 41 users, and was generously provided by [1, 2]. The authors of [1, 2] also generously provided code to extract features from the datase, this code was used with very few modifications. A raw stroke is a time seperated series containing time, x position, y position, phone orientation, pressure, finger area, and finger orientation. Detailed information on the raw sensor data, how the touch information was collected, and the 31 features that are extracted can be found in [1, 2].

2.2 Normalizing

We normalize the data as follows.
Let $F \in \mathbb{R}^{mxn}$ be our matrix of feature values given from the training data, where $f_j^{(i)} \in F$ is the j^{th} feature of the i^{th} example.
We define:
$\sigma_k = standard\ deviation(\{f_k^{(i)} : 1 \leq i \leq m\})$
$\mu_k = mean(\{f_k^{(i)} : 1 \leq i \leq m\})$
Our normalized matrix of feature values $N \in \mathbb{R}^{mxn}$ is defined so that the j^{th} feature of the i^{th} entry, $n_j^{(i)}$ equals the following: $n_j^{(i)} = \frac{f_j^{(i)} - \mu_j}{\sigma_j}$

3 The Classifiers

3.1 kNN

One of the classifiers is a kNN classifier. The classifier receives a raw stroke, transforms it in to a normalized feature vector, and then finds the closest feature vector in its database. Closeness is defined in terms of the Hamming distance. The classifier only looks for the closest neighbor (i.e. k = 1), and labels the stroke as the user with the closest feature.

3.2 SVM

Many people use binary supporting vector machines with the notion of 1 user to all of the other users. The classifier used here is a multiclass SVM with a gaussian kernel. Intuitively, an SVM finds the best finite width walls to divide the data into each class, and the gaussian 'kernel trick' allows the walls to be curvy without adding much computational overhead. More formal definitions can be found easily.

4 The Attack

The attack is based on the idea that the classifiers rely on a notion of closeness between a user's feature vectors.

The adversary has a database of feature vectors from an arbitrary group of users, A, with the corresponding raw touch gestures; perhaps the adversary created a mobile application to collect the touch gestures. The kNN classifier, which can be thought of as an authentication server, has an independent set

of feature vectors, C. Now, we assume the adversary gets a full copy, or a partial copy, of the kNN classifier's feature database, but the adversary does not have the classifier's raw stroke data. The attack uses the compromised features, C, and the adversary's feature database, A, to forge raw strokes and pose as any given user.

4.1 Rotation and a Thorough Description of the Attack

First, the adversary removes the following features: starting x-position, starting y-position, ending x-position, and the ending y-position. Now, for a particular user, the adversary takes each of ther user's compromised features, $u^{(i)} \in C$, and the adversary finds $a^{(k)}$ such that $k = \min_k(||u^{(i)} - a^{(k)}||_2)$, where $a^{(k)}$ is the k^{th} example in A, the adversary's feature database. The adversary then takes the stroke, $s^{(k)}$, corresponding to $a^{(k)}$ and transforms the stroke to more closely resmble the stroke represented by $u^{(i)}$, giving us $\hat{s}^{(i)}$. The feature $u^{(i)}$ contains the starting and stopping location for its gesture, we use this information to transform $s^{(k)}$ in to $\hat{s}^{(i)}$ as follows:

Algorithm 1 Stroke Matcher

procedure ShiftAndRotate
  for i := 1 to length(u) **do**
    $n := length(\hat{s}[k].x)$
    $\delta_x := s[k].x_1 + u[i].startingX$
    $\delta_y := s[k].y_1 + u[i].startingY$
    $\hat{s}[i].x := s[k].x - ones(n, 1) * \delta_x$
    $\hat{s}[i].y := s[k].y - ones(n, 1) * \delta_y$
  for i := 1 to length(u) **do**
    $y_1 := \hat{s}[i].y_{end} - \hat{s}[i].y_1$
    $x_1 := \hat{s}[i].x_{end} - \hat{s}[i].x_1$
    $\omega := atan2(y_1, x_1)$
    $y_2 := u[i].endingY - u[i].startingY$
    $x_2 := u[i].endingX - u[i].startingX$
    $\phi := atan2(y_2, x_2)$
    $\theta := \phi - \omega$
    for j := 1 to length($\hat{s}[i].x$) **do**
      $\delta_x := \hat{s}[i].x_j - \hat{s}[i].x_1$
      $\delta_y := \hat{s}[i].y_j - \hat{s}[i].y_1$
      $\hat{s}[i].x_j := cos(\theta) * \delta_x - sin(\theta) * \delta_y$
      $\hat{s}[i].y_j := sin(\theta) * \delta_x + cos(\theta) * \delta_y$
      $\hat{s}[i].x_j := \hat{s}[i].x_j + \hat{s}[i].x_1$
      $\hat{s}[i].y_j := \hat{s}[i].y_j + \hat{s}[i].y_1$

The intuition is that the algorithm changes $\hat{s}^{(i)}$'s starting location to be the same $u^{(i)}$'s starting location and then rotates $\hat{s}^{(i)}$ so that its end-to-end line is colinear with $u^{(i)}$'s stroke's end-to-end line. An illustration of this process follows.

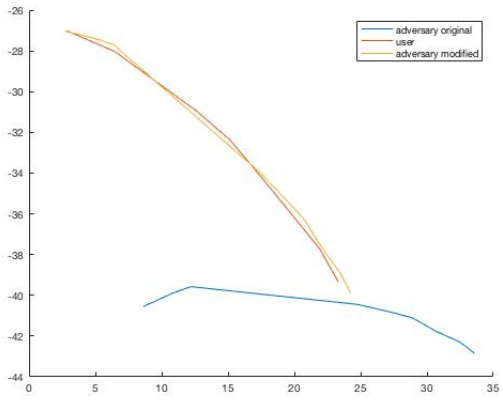


Figure 1

The adversary then sends $\hat{s^{(i)}}$ to the server which translates it to the feature space, giving us $\hat{u}^{(i)}$. The goal of the adversary is to minimize $||\hat{u}^{(i)} - u^{(i)}||_2$.

4.2 Midpoint Adjustment

There are three features which are equal to the median of raw sensor data: mid-stroke pressure, mid-stroke area covered, and mid-stroke finger orientation. Similar to the rotation in 4.1, we simply increase all of the pressure, area, and finger orientation sensor captures for a stroke until their median values equal the median values from the compromised features. Additionally, we remove those features from the adversary's kNN matching criteria when performing this manipulation.

5 Results

5.1 The Classifier

Half of the data, about 10,000 strokes, was used for the training set. The other half of the data was used for the test set. Experiments showed that using Hamming distance for the kNN, as opposed to euclidean distance, lead to more accurate classifications on the training set. Similarly, the Hamming distance performed well on the test set, suggesting this generalizes to the data well. Not surprisingly, normalizing the data also dramatically increased the performance of the SVM and the kNN classifiers. While I think more work is needed to train the SVM's hyperparameters, I was able to improve the performance of the exclusive-SVM by removing features which were found using forward search on the training data. Improvements to the classification precision on the test

set are shown below. The percentages in tables 1 and 2 are averaged over all users in the database.

classifier	1	2	3
exclusive-kNN	.0912%	51.71%	56.32%
overlap-kNN	100%	100%	100%

Table 1

1: Feature vectors are not normalized and euclidean distance is used
2: Features vectors are normalized and euclidean distance is used
3: Feature vectors are normalized and hamming distance is used

classifier	1	2
exclusive-SVM	29.85%	41.26%
overlap-SVM	98.93%	97.34%

Table 2

1: SVMs with gaussian kernels with 31 features
2: SVMs with gaussian kernels with 20 features

The **'exclusive'** performance, is analgous to the real-world scenario, it is the percentage of a user's new strokes which are classified as said user, using the user's old strokes as training data. The **'overlap'** performance is the percentage of a user's strokes classified correctly when the classifier is trained with data containing said strokes. While the 'overlap' performance is not analogous to a real-world scenario, it is useful when assessing how well the attack generates the desired strokes, which is the ultimate goal of this article. Additionally, if the classifier does not refresh its training data, then the 'overlap' performance of the attack becomes the real-world performance.
The figure below shows the precision of the classifiers on the 22 users in the test set.

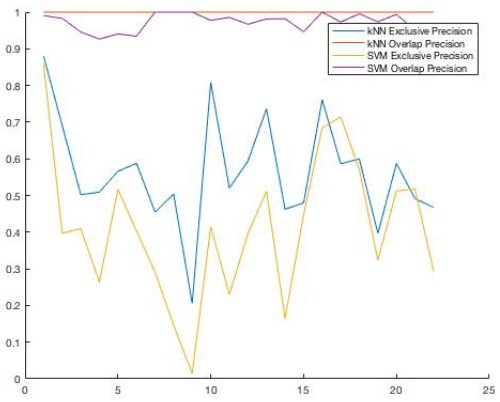
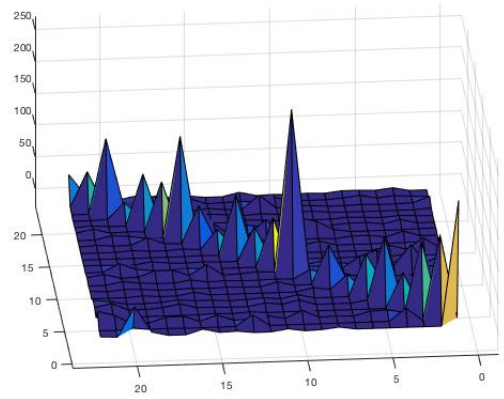
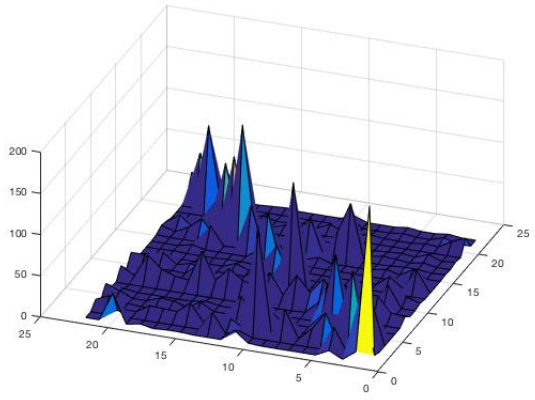


Figure 2

Interestingly, the SVM and kNN distributions follow each other in the exclusive case. This suggests that some users are more difficult to classify than others, or the classifiers need more training data for some of the users. The figures below show what each user's strokes are classified as. The horizontal axis represents the estimation and the axis going in to the page represents the true classification. The diagonal represents strokes being matched to the correct user.



Confusion Matrix for exclusive-kNN-3.
Strokes from 22/22 users were matched to the correct user more than any other user.



Confusion Matrix for exclusive-SVM-2.
Strokes from 19/22 users were matched to the correct user more than any other user.

While the precisions are not perfect, the confusion matrix for the exclusive-kNN-3 suggests classifying multiple strokes at a time can provide near perfect accuracy!

5.2 Principal Component Analysis

Additionally, principal component analysis was used. Unfortunately, PCA did not improve the performance

of the classifiers. However, we were able to retain roughly the same precision with a reduced feature set, which can improve the computational performance of the classifiers and reduce training time. The principal component feature set was not used while evaluating the classifiers nor was it used while evaluating the attack. Results from the test set are given below. Note: we are using the kNN classifier, with euclidean distance, and exclusive datasets.

Principal Comopnents	Exclusive Precision
5	17.09%
10	38.02%
15	49.36%
20	51.52%
25	51.98%
30	51.71%

Table 3

With only 15 features we can achieve a precision of 49.36%, compared to a precision of 51.71% (from table 1) with the original 31 features.

5.3 The Attack

The attack was trained on the kNN classifier. Attacking the SVM classifier demonstrates how well the attack generalizes.

While experimenting, it was discovered that using the compromised features to rotate the adversary's strokes, as decribed in section 4.1, improved the precision of the attack on two users. Similar results occurred with the the full data set. Similarly, using the compromised features to adjust the adversary's pressure, finger area, and finger orientation, as described in section 4.2, also improved the precision of the attack. Less surprisingly, normalizing the data also dramatically improved the precision of the attack. The table belows shows how often the adversary's generated strokes were matched to the desired user, averaged over all of the users.

classifier	1	2	3
exclusive-kNN	41.98%	49.95%	58.09%
overlap-kNN	50.71%	95.34%	99.71%
exclusive-SVM	21.13%	27.20%	31.59%
overlap-SVM	41.23%	65.02%	79.72%

Table 4

1: Strokes are not rotated and the pressure, area, and finger orientation are not adjusted
2: Strokes are rotated, but the pressure, area, and finger orientation are not adjusted
3: Strokes are rotated and the pressure, area and finger orientation are adjusted
Note: mimicked features are removed from the adversary's kNN search (i.e. when rotating we remove the starting and ending positions from the adversary's kNN match criteria). Also, in this context 'overlap' refers to when the classifier's training data contains the compromised strokes which the adversary is counterfeiting, and 'exclusive' means the classifiers are trained on the same data minus the counterfeited strokes.
Unfortunately the classifiers are far from ideal, so let's look at the relative precision.

classifier	relative precision
exclusive-kNN	1.031
overlap-kNN	.9971
exclusive-SVM	.7656
overlap-SVM	.8190

Table 5

The 'relative precisions' for the kNN classifiers are computed by dividing column 3 in table 4 by column 3 in table 1. The 'relative precisions' for the SVM classifiers are computed by dividing column 3 in table 4 by column 2 in table 2. Intuitively, the 'relative precision' tells us how precise the counterfeited strokes are compared to the true user's strokes. The table above suggests that the attack is overfit to the kNN classifier, but still generalizes well to the SVM. The 'relative percisions' above suggest the counterfeited strokes may fall well within an authentication server's tolerance for error.

6 Conclusion

While the kNN classifier performs well enough to be used in a real-world authentication server, the SVM classifier may lead to an intolerable number of misclassifications. Notably, figure 2 suggests we may be able to mitigate these issues and greatly improve the performance of the SVM classifier, and the kNN classifier, by collecting more data for users that were difficult to classify. On the other hand, both classifiers were very vulnerable to the kNN attack. Given that an authentication server would need a tolerance for error, the adversary's conterfeit strokes are very likely to succeed. The SVM classifier seemed less susceptible to the attack, but it is more likely that the attack is overfitted to the kNN database. Regardless, the attack seems to generalize well. Clearly, both classifiers would strongly benefit from securing their feature databses with something analgous to the hashing

of passwords. Moreover, authentication servers using machine learning algorithms must constantly refresh their feature databases to defend against counterfeit attacks. In conclusion, touch gesture based authentication is a reasonable form of secondary authentication, but there are many security concerns that need to be addressed before these authentication mechanisms are used to protect private information.

7 References

[1] Neil Zhenqiang Gong, Mathias Payer, Reza Moazzezi, Mario Frank, "Forgery-Resistant Touch-based Authentication on Mobile Devices", ASIA CCS 16,
http://www.mariofrank.net/touchalytics/index.html
[2] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song. Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication. IEEE Transactions on Information Forensics and Security, 8(1):136148, 2013.
http://www.mariofrank.net/touchalytics/index.html
[3]S. Bleha, C. Slivinsky and B. Hussien, "Computer-access security systems using keystroke dynamics," in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 12, no. 12, pp. 1217-1222, Dec 1990.
URL: http://ieeexplore.ieee.org/stamp/
stamp.jsp?tp=&arnumber=62613&isnumber=2279