CS 263 Final Project
Fall 2016

Accelerating Good (and Evil): Using Accelerometer Data to Catch Spam (and Spy on People)

Ezra Zigmond, Ankit Gupta, Zack Chauvin

Abstract
To prevent automated systems from accessing certain webpages pages, website developers often add techniques to verify that a user is a human. These techniques generally involve tasks that are relatively easy for humans but difficult for automated systems. One of the most popular such methods is CAPTCHA. However, these tasks can still be difficult or unpleasant to complete for humans. In general, they can reduce the quality of user experience, which may make a user frustrated when interacting with a system, particularly for those with disabilities.

To address these issues, we explore a replacement for CAPTCHA that clandestinely uses a mobile client's motion data to determine whether the client is an automated system. This motion data can be collected through Safari and Chrome mobile browsers on iOS and Android without any user permission. We analyze techniques that adversaries could use to defeat the classification ability of such a system, and propose and implement means to prevent such adversaries. Finally, we compare our results to other methods for simplifying the user verification pipeline, notably reCAPTCHA by Google.

Ultimately, we find that the model is robust against a variety of threats and believe that our technique can significantly improve the user experience, particularly for those with learning disabilities. However, this model does still contain vulnerabilities that allow adept attackers to subvert the system. Thus, further development is needed before this technique can be widely deployed.

Contents

1. Introduction

Web sites often wish to distinguish page access by human users and by automated bots. For example, an online vendor may wish to prevent ticket scalpers from using scripts to immediately purchase tickets and re-sell them at a much higher price. Thus, over time, web programmers have deployed a variety of means to prevent unauthorized access by automated systems into web services. A class of solutions for this is called CAPTCHA [1], and the most widely used CAPTCHA is reCAPTCHA by Google [2].

reCAPTCHA is a system created to distinguish between humans and bots. The original reCAPTCHAs presented a short string of slightly obfuscated text in image form, and the user was asked to write what was in the image in a text box to verify their identity. However, these images are difficult to parse and slow down the process of trying to access a desired resource.

Furthermore, modern machine learning techniques for image recognition can reliably solve reCAPTCHA, reporting accuracy levels above 99%[2]. As such, Google re-engineered reCAPTCHA to increase the difficulty of the task and incorporate other techniques.

Currently reCAPTCHA uses a variety of methods to attempt to verify that a user is a human. For example, reCAPTCHA may present a series of images, and ask the user which of those images contains a certain object, such as a particular animal. This is a task that should be relatively easy for a human to solve, but can be cumbersome, as we discuss in greater length in Section 2.2. This was also difficult for bots to defeat in the past, but recent demonstrations have shown a mix of reverse image search and deep learning techniques for image classification can quickly and accurately solve image-based CAPTCHAS [3].

However, the current reCAPTCHA system does not always require a user to solve an image task. Much of the time, the user can just click a single button that says “I’m not a robot”. In the background, it is thought that this system works by tracking a user’s mouse movements and scrolling behavior [4] to determine whether the behavior appears to be that of a human. If so, no image challenge is presented. If the behavior seems unusual, an image challenge is presented, some of which is solvable by the aforementioned techniques.

This shift to incorporate more hidden elements like mouse movement has the potential to make CAPTCHAs easier and faster for humans. However, the mouse movement framework has the downside that it relies on desktop use, since similar tracking of the mouse isn’t applicable to mobile phones. Because of this, mobile versions of reCAPTCHA must rely on other sources of information to judge whether or not a user is a human.

In this paper, we present a mobile-friendly CAPTCHA system that is empirically difficult to thwart. The system is based on our hypothesis that accelerometer data which tracks the way that a user holds a phone as they engage with the CAPTCHA can be used to differentiate between human and robot agents, analogous to mouse movement on desktop. The creation of a CAPTCHA system which uses accelerometer data instead of challenges will allow mobile users to more easily and quickly prove that they are humans. As Internet access becomes increasingly prevalent on mobile devices, an accelerometer-based CAPTCHA system will improve the browsing experience while continuing to deny bots access to protected resources.

While our research on accelerometer tracking for mobile has the potential to be beneficial for the browsing experience, we also consider the potential for abuse of this data. In addition to evaluating the effectiveness of this approach for CAPTCHAs, we also explore what can be learned by having accelerometer data for users that are attempting to access certain pages. We evaluate the ability of the service to tell what type of activity the user is engaged in while completing the CAPTCHA, whether it be walking, sitting, or riding in a car. This information could be useful to advertisers who are interested in activity details about specific consumers. This use of information may be considered unacceptable by users of the CAPTCHA and must therefore be explored to understand whether or not this technology could be deployed in practice.

Our contribution in this paper is three-fold:

1. We present a motion-based CAPTCHA challenge and evaluate its effectiveness at catching non-human users.
2. We describe the design of a CAPTCHA server implementation and consider its security properties.
3. We demonstrate that a malicious company providing a motion-based CAPTCHA service could easily use motion data to track user behavior with high accuracy.

2. Related Work

2.1 CAPTCHAs and Attacks on CAPTCHAs

Our research builds upon previous work in the bot detection space. One area of research has covered the variety of options that are available to services implementing a CAPTCHA. The Asirra system from Microsoft Research documents an innovation to CAPTCHA systems that previously used text-based challenges. Their implementation uses imaged-based challenges requiring users to distinguish one class of photos from another in an attempt to beat bots that are unable to do this difficult visual classification [5].

Even within the space of image-based CAPTCHAs, there has been a lot of research concerning the many possible types of challenges that could be used to identify bots. In their paper, Chew and Tygar detail three image challenges concerning naming, distinguishing and identifying [6]. Another possible challenge is identifying which image is oriented correctly, as was explored in [7].

While these challenges have the advantage that they are easier for human eyes and harder for bots to beat using computer vision than text-based challenges[7], they are still a nuisance for users. This has lead to the exploration of techniques that do not require explicit challenges, such as Google's new reCAPTCHA. This system reduces the strain on users by not always requiring a challenge. There are currently no papers either from Google or third parties detailing how this system functions, but interviews with Google employees indicate that Google uses mouse movement data of agents that interact with the reCAPTCHA to determine if they are humans or bots [4].

The work on alternatives to classic CAPTCHAs was beneficial both because it reduces the burden on the user trying to prove their humanity, but also because text and image challenges are becoming more vulnerable to machine learning attacks. For instance, a research group was able to achieve 82.7% accuracy on the Asirra challenge [8]. Similarly, another paper manages to cause the Google reCAPTCHA to provide an image challenge, and then uses visual learning techniques to solve the image challenge with 70.78% accuracy [3]. Both of these papers reveal that current CAPTCHA systems are vulnerable to learning techniques and motivate exploring alternative methods of bot detection.

2.2 Usability Limitations of CAPTCHAs

CAPTCHAs present significant usability hurdles that can worsen user experience. Users report widespread frustration with CAPTCHA services and that users with learning disabilities experience even greater frustration than users without such disabilities [9]. Other research has suggested that despite the designers of CAPTCHAs generally believing that they should be easy for humans to solve, people routinely have a difficult time solving them [10].

It should be noted that Google has improved its reCAPTCHA system over time, and it itself has claimed that the primary reason for this was to keep preventing bots while improving the user experience, ostensibly in reaction to some of the aforementioned research [11].

Furthermore, Google has now indicated that they are launching a new version of reCAPTCHA this year that is mysteriously called "Invisible reCAPTCHA" [12], which they claim will work on both desktop and mobile clients. While the mouse-tracking system discussed before along with browser fingerprinting gives ample means to implement such a system on desktop, the relative lack of access to input sources on mobile devices makes the data sources on such a system less clear for mobile devices. As such, we intend to investigate the feasibility of using accelerometer data as a data source for such a system.

2.3 Surveillance via Accelerometer Data

As we explore the possibility of collecting accelerometer data for CAPTCHAs, we must also consider what can be learned from this data to evaluate the potential of malicious surveillance. Research has also been done in this field, with a study achieving 92% accuracy on classifying a large amount of accelerometer data collected from mobile devices as jogging, walking, standing, stair climbing and sitting. [13] This research shows that if a service were to collect accelerometer data for the purpose of determining whether an agent is a human or a bot, it could also use the data to determine what the user is doing while interacting with the CAPTCHA. Our work expands on this research by attempting to accomplish similarly accurate classification of accelerometer data while not relying on a huge amount of data or complex models which are two traits of the prior research. We show in our results that even with small amounts of unlabeled accelerometer data, we can predict user behavior with high accuracy.

3. Design of CAPTCHA Challenge

3.1 Threat Model

A CAPTCHA server has a classifier, described in the next section, that has been trained to distinguish between human and non-human data. For the purposes of this paper, we will refer to non-human data interchangeably as robot or bot data.

We are assuming that the adversary has not compromised the server and stolen the actual model. However, we are assuming that the adversary has reverse engineered the protocol used by client sites to send non-malicious user data to the server and can send a large number of requests to the server at any given time point, such as through a botnet. The adversary can also successfully spoof its user agent headers that it appears all of the requests are coming from a mobile device. We assume that the server can handle a large number of requests, and has all of the usual protections from denial of service attacks.

Furthermore, we assume that the adversary knows that the server generally uses the classification architecture described in the next section, and can relatively easily write a similar architecture on its own. However, we assume that the adversary has substantially fewer samples than the creator of the CAPTHCA server. For example, the server may be run by a large technology company, who can track a few seconds of motion data from its millions of users, while the adversary cannot access nearly the same volume of information quickly. This means that it would be impractical for an adversary to merely collect legitimate data from humans and send it to the server. Instead, the adversary must use whatever data it has collected to generate large amounts of fake data to try to trick the server. We do not aim to protect against the case where an attacker can easily collect human motion data (for example, if the attacker were the owner of a large social media site used on mobile phones).

3.2 High-Level Learning Framework

Fundamentally, our CAPTCHA challenge is a classification problem, where the CAPTCHA server judges whether an input sequence is a human or is machine-generated. Thus, the input is a sequence (not necessarily of fixed size) of data from a user's accelerometer, and the classifier must determine whether this sequence corresponds to a human-like sample or a robot-like sample. The fundamental hypothesis we have is that it is difficult to create large numbers of human-like samples without having to manually collect the data, which is slow and expensive. However, it should be relatively easy to train an algorithm that can determine whether a particular sample is a human or not.

Furthermore, we wished to build a learning framework without any pre-engineered features. Past efforts at using accelerometer data to characterize user behavior have involved this type of manual feature engineering. However, this presents a security risk, as more obvious features will be easier for an attacker to engineer to defeat our classifier. Moreover, not manually engineering features prevents us from encoding our own biases about the important aspects of the accelerometer.

In accordance with this, we used an acceptor recurrent neural network (RNN), written in Tensorflow (`https://www.tensorflow.org/`), to write our classifier. An acceptor RNN is a model that takes sequential data as input. Then, the RNN cell loops over the data, and at each step, applies a series of linear and non-linear transformations in order to update an internal fixed-size state vector and outputs a fixed-size vector of size H. After looping over the full sequence, the final output vector offers a fixed-size encoding of the entire sequence, and can be fed into any linear classification model, which in our case is a 1 hidden-layer feed-forward neural network.

Let M be the number of time steps in a user-supplied sample, and K be the number of variables output by the accelerometer. Additionally, let there be N samples to train on.

In our case, the input is a vector of shape N x M x K, where each sample is thus a vector of shape M x K. The RNN will loop over the M steps of a particular sample, and output a vector of size H, which will be the input to the linear classifier. The classifier will emit the probability of the sample being human-supplied, and to make decisions, we will pick human if the probability is more than .5.

3.3 Collecting Human Examples

3.3.1 Surveying Users

In order to train our model, we surveyed Harvard College undergraduates to collect motion tracking information. We solicited users via Harvard mailing lists, indicating to users that we would capture 4 seconds of motion information when they clicked a button on a web-page we developed. This webpage remains hosted at `cs263.herokuapp.com`.

From these users, we obtained 229 samples of motion data from a variety of conditions, as we did not specify to users any particular way that they should hold their phones.

Each of these data samples was of size M x K, where M is the number of time steps, and K = 6, the number of pieces of data that the accelerometer emitted. Accelerometer data was collected by logging data from the JavaScript DeviceMotionEvent[14]. At each of the M time steps that the event fired, we recorded six values: the X, Y, and Z accelerations in m/s^2 and the X, Y, and Z rotation of the device in degrees (referred to in the documentation and in our code as alpha, beta, and gamma respectively).

3.3.2 Activity-specific Data

Afterward, we decided it would be good to supplement the data we collected with data specific to certain activities. Since we did not trust surveyed individuals to reliably indicate their actions at the time of visiting the webpage, we collected this data ourselves. We created a second webpage at `cs263.herokuapp.com/experiment`, where we manually collected data while conducting the following activities: walking, sitting, going up stairs, being in a car, and leaving the phone on a table. This generated an additional 200 samples that were of sufficient length.

These samples were then used for our various classification tasks.

3.4 Generating Adversarial Examples

In this section, we describe our method of generating adversarial examples. In particular, our threat model involves an adversary that will try to beat our classifier by presenting computer-generated data, which may have been generated using a sample of real human data. To prevent this, we generated our own non-human data and trained the classifier to recognize it as not human-like. The following subsections survey the various techniques we used to generate this data.

3.4.1 Timestep Sampling

For each of the M x K different values in the human examples, we calculate the mean and variance of that value across all human examples. We then generate adversarial examples by sampling each value from a normal distribution defined by the calculated mean and variance for that value.

3.4.2 Difference Sampling

The problem with the timestep approach alone is that it ignores that the acceleration of a user's device is likely to be correlated across timesteps (a user is relatively unlikely to jerk their phone abruptly between two timesteps). Therefore, we generated adversarial examples that attempt to mimic human changes between timesteps. To this end, for each of the (M - 1) x K pairs of adjacent sequential points, we calculate the mean and variance of the difference between the second point and the first point. We then generated adversarial examples where the first timestep is generated as above (based on the mean and variance of the points in the first timestep) and the remaining M - 1 timesteps are generated by adding differences sampled from the calculated means and variances to the previous timestep.

3.4.3 Random Noise

To teach the model to ignore completely random values, we generated examples that look like random noise. Based on the minimum and maximum values across all human examples for each of the M x K different values, we generate adversarial examples by sampling each value uniformly between the minimum and maximum for that value in the human data.

3.4.4 0-centered Gaussian Noise

An attacker might take a legitimate human sample and add a small amount of 0-centered noise to it to avoid try to avoid a naive replay-prevention system that only detects sending exact duplicates of previous data. To generate the adversarial data, we picked a random sample from the legitimate human data and for each of the M x K values in the sample added noise sampled from a normal distribution centered at 0 with variance equal to the signal-to-noise ratio of the value in the human data.

3.4.5 Hidden Markov Model

Finally, we generated a form of adversarial data representing an attacker who came up with a new form of data that we had not anticipated. To this end, we trained a five-state hidden Markov model (HMM) on human data using the Python hmm-learn library (`http://hmmlearn.readthedocs.io/en/latest/`). We then sampled sequences from the trained HMM model to generate adversarial data.

4. Design of CAPTCHA Server

We now discuss the design and implementation of the centralized CAPTCHA server that serves the motion classification models and allows client servers to authenticate users. From the perspective of a client website wishing to use the service, the process is very similar to using Google's reCAPTCHA. (In the following sections, *client* refers to someone who signs up for our service and wishes to verify that visitors to their website(s) are human. A *user* refers to someone trying to access the client's site).

4.1 Client Key Management

Logging in: In order to use the CAPTCHA service, a client must log into a valid Google account using Google's OAuth2 flow. We require users to log in to manage their client keys for three primary reasons: first so that if a user forgets their client credentials, they can log into the CAPTCHA server to find the information, and second so that we can place a per-account limit on the number of client credentials (to prevent an easy DOS attack where a malicious client repeatedly requests new keys in an attempt to consume disk space on the server or occupy the server in creating keys) and finally so that abuses of a particular client key could be traced back to a particular user (this feature could be useful for example to contact a user with an alert if their website is receiving a spike in spam traffic or to ban the user if network logs seem to suggest they are using their client keys to attempt to break the CAPTCHA model). We chose to require sign-in through a Google account since presumably Google has existing security measures in place to prevent malicious account sign-ups which makes it more difficult for a banned user to simply sign up for another account.

Managing keys: A client who has successfully authenticated with a Google account can view a list of their created client keys, and create new credentials (up to a limit set by the server). A client may wish to create multiple keys in case they run multiple websites and want separate keys for each site to minimize the potential damage if one key is compromised. Logged-in clients can also revoke their own keys.

A client key consists of two parts: a site ID which publicly identifies the client and a client secret which should only be known by the client and the CAPTCHA server. Both the site ID and the site secret are 40-byte ASCII strings (we chose this length based on the length of the client secrets given out by Google's reCAPTCHA) generated by using the secure PRNG in the Python Cryptography Toolkit (https://www.dlitz.net/software/pycrypto/). The site ID is necessarily unique across all client keys but multiple clients may have the same site secret with very low probability.

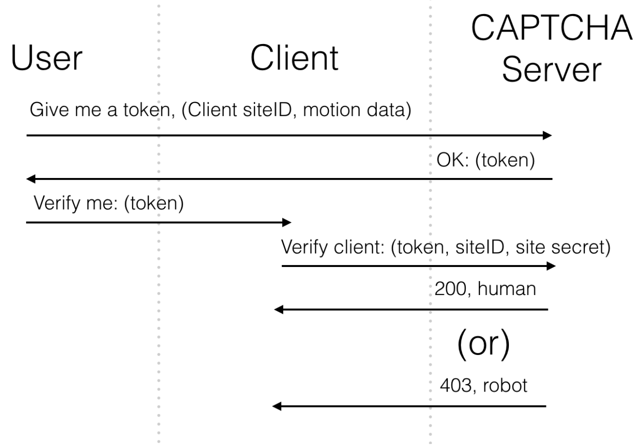

User
Client
CAPTCHA Server
Give me a token, (Client siteID, motion data)
OK: (token)
Verify me: (token)
Verify client: (token, siteID, site secret)
200, human
(or)
403, robot

Figure 1. Network diagram of steps to verify a user as human.

4.2 User Verification

We now detail the steps for a user wishing to access a sensitive part of a client website where the client wants only human users (for example, account creation), so the client can verify that they are a human user. Figure 1 summarizes the network messages exchanged between the user, client, and server as part of the verification process.

User requests a token : A user sends an HTTP POST request to the CAPTCHA server containing the site ID of the client site they wish to access and a 260 by 6 array of floats which corresponds to 4 seconds of JavaScript motion events. The CAPTCHA server checks that the site ID is valid and returns a 400 error if an invalid site ID is given. Otherwise, the CAPTCHA server returns a token to the user. A token is a random 420-byte ASCII string (this length was chosen by observing the length of tokens generated by Google's reCAPTCHA). Note that a token is returned to the user whether or not the CAPTCHA server verifies the motion data as human, making it more difficult for the user to attempt to use the CAPTCHA server as an oracle to defeat the verification model. Internally, before returning the token to the user, the CAPTCHA server verifies whether or not the user is human.

If the user is verified as human, the CAPTCHA server assigns a timestamp to the token and stores it in a database that maps site IDs to valid user tokens for that site. If the server identifies the user as being non-human, the token is discarded.

Client verifies the user: A user can then present the token received from the CAPTCHA server to the client. The client sends the token, its site ID, and its site secret to the CAPTCHA server. If the site secret is invalid for the site ID, a 403 error is returned. We require that the site secret be sent to verify a token to prevent users from trivially checking if the token they received is valid infeasible and to prevent clients from searching for valid tokens. If the token submitted by the client is valid for the site and has not expired, a 200 response is returned to the client indicating that the user has verified themselves as human. If the token is expired (as measured by a server-defined token lifetime) or if the token is invalid for the site, a 403 error is returned (the error message indicates whether the token was expired or invalid in case a client wishes to treat these cases differently. If the server responds that the user is human, the client can then allow the user to perform the requested action with high confidence that the user is human. If the server indicates that the user is non-human, the client could choose to reject the user outright or to present a different CAPTCHA-style challenge.

4.3 Integrating with a Client

Integrating the CAPTCHA system with a client's existing website should be no more difficult than integrating with Google's reCAPTCHA scheme. The client could embed a JavaScript from a CDN run by the CAPTCHA service. The client can then place an HTML div with the id ”captcha” and the JavaScript code will populate the div with an ”I am not a robot” button. When the user clicks the button, the client JavaScript will collect 4 seconds of motion data from the user and then complete the token-request process described above. The JavaScript will then populate a field in the div with the received token which can be submitted as form data to the client server. Note that this could also be an invisible div that is passively capturing accelerometer data, and that sends it for verification at some time without informing the user.

The JavaScript would also need to detect (presumably based on user-agent headers) whether the user is accessing the client site through a mobile browser and only present the motion-based CAPTCHA in this case. If the user is on a desktop device, the client could then choose a fallback image- or text-based CAPTCHA to present.

The only server-side modification necessary on the part of the client is verifying the submitted token before allowing a user to perform a sensitive action. While we have not implemented the bespoke JavaScript necessary for this sort of integration as part of our prototype, it could easily be achieved by modifying the code used for our data collection server.

4.4 Security

We now consider potential security risks to the implementation of the CAPTCHA server. Note that this does not exhaustively consider attacks on any CAPTCHA server (for example, simple DoS attacks by repeatedly asking the server to verify invalid tokens), but focuses on non-obvious vulnerabilities with our model.

Replay Prevention One of the central risks is that if our classifier does not work perfectly, an adversary may eventually determine a sequence of inputs that is incorrectly classified as human, and then repeatedly use that. Even worse, an attacker can simple record herself holding a mobile device, and then use that manually-collected data, or small perturbations of it, as inputs to the classifiers.

One potential solution to this is to have a static list of past inputs (or hashes of them) over some bounded amount of time. However, an attacker can easily thwart this attack by adding small zero-centered Gaussian noise with low variance to each data sample, which will be small enough to note change the result of our classification model, but different enough to lead to a different stored hash.

As such, we develop a binning model to reduce the impact of added noise. In particular, rather than storing a hash of the entire sequence, we first take the mean of the sequence over time. Thus, we take our input sequence of size M x K and take the mean over M, giving a one-dimensional vector of size K. Then, we break each element of the vector K into D bins, where D is a configurable parameter, and where the bins were developed by looking at the maximum and minimum possible values for each of the K variables emitted from the accelerometer.

Then, we place each of the vector elements into the appropriate bin (between 0 and D), and hash the resultant vector. Since D is configurable, we are able to modulate how much variance we find acceptable without considering the input to be different. In particular, if a user now adds small noise to human samples, this hashing strategy will lead to similar but different samples to hash to the same value, which we can then reject. We use the built-in Python hash function - naturally, this hash function can be replaced with any hashing algorithm.

If the user uses very different samples, then our classifier will no longer view it as human-like, and will reject it. Note that this also naturally prevents replay attacks, since using the sample input will quickly hash to the same value as a previous one.

Adversarial Machine Learning Attacks Closely related to replay attacks are a class of attacks related to the fact that we are using machine learning models whose inner function is generally obfuscated to the implementer, and whose decision-making is not always obvious to the programmer that designed them. As a result, there is considerable literature about various techniques that have generated adversarial examples that can trick machine learning models into making incorrect predictions.

For example, in [15], the authors show how adding a small amount of noise that is imperceptible to human eyes to an image can reliably fool a classifier that correctly classifies images to instead classify all of them as ostriches. Fascinatingly, a human would see no differences in these images, but small changes in pixel density can propagate errors through the model and ultimately lead to a significantly different result.

The techniques we described in 4.4 are designed to prevent this type of attack, since the amount of noise added should be small enough to prevent adversarial noise added to fool the machine learning models, but generally too little noise to fool the binning technique.

Timing Attacks: When a user asks the CAPTCHA server for a new token, the server always generates a token and validates the data sent by the user but only stores the token in a database if the token is deemed valid. Depending on the particular implementation of the server design, it may be feasible to measure this timing difference and enable the client to use the CAPTCHA server as an oracle to try to defeat the model. If this is of particular concern, an implementation could modify the server so that it always stores the token regardless of whether or not it is valid, along with a tag indicating the validity of the token. This would eliminate the timing channel attack but would enable a DoS attack in which a malicious user floods the server with junk data, forcing it to generate and store many tokens.

Token Harvesting: Unlike Google's reCAPTCHA, our server implementation does not restrict on which domains a user is allowed to complete the challenge for a particular site ID. This is because we were unable to find a satisfactory fix to the exploit in Google's site restriction identified in Sivakorn [3]. Therefore our CAPTCHA server implementation is vulnerable to the token harvesting described by Sivakorn [3]. The appeal of this token harvesting is that it allows a malicious user to generate many valid CAPTCHA tokens while bypassing potential anti-hammering protections of the client server. However, we attempt to limit the feasibility of such schemes by adding a timestamp to each token and considering an otherwise valid token to be expired if its timestamp is too old. If the server sets the lifespan of a token to be sufficiently small, we believe offline token harvesting is less appealing because an attacker cannot collect and sell tokens in advance, but rather would have to generate tokens offline and use them immediately. (Note that Sivakorn [3] confusingly uses the term token harvesting to refer to harvesting *cookies* that are used by Google's risk-analysis model, which is not applicable to our design.)

5. Evaluation

5.1 Evaluation Methods

In this section, we describe the methods we used to evaluate the efficacy of our approach. We both evaluate the effectiveness of the classifiers at classifying what they are intended to model, and at resisting various forms of malicious attacks.

Note that we do not claim to have exhausted all of the possible strategies that an adversary can use. However, we do try a variety of methods and report our performance against them broadly.

In general, we generated independent training and test sets using the methods below. Both of these sets were made by splitting the human data into 80% training and 20% testing, and independently generating non-human data by the methods described. We evaluated the performance of the trained classifier by testing it on the test dataset and calculating the classification accuracy.

5.1.1 Human vs Not-Human

The first classifier we trained was designed to distinguish between human samples and non-human samples. To train this classifier, for the human-samples, we used the data collected from surveyed users and ourselves through the methods that we described earlier. For the activity-based data, we stripped the activity that the person was doing for this classification task, since the goal was just to determine whether or not the data came from a human.

The difficult part in this case was to generate the negative samples, meaning examples of non-human data. Section 3.4 describes a variety of methods for creating these adversarial examples. To actually test the effectiveness of these various methods, we trained one classifier with negative samples from each of these methods, along with one classifier trained with a mix of these methods. We originally did not include the hidden Markov model (HMM) generated data in the mixed model to see how well the mixed model would perform against previously unseen types of data. However, upon observing that the mixed model performed extremely poorly (around 11% accuracy) against HMM-generated data, we trained a second mixed model that also included HMM-generated data.

Furthermore, when we tested our model, we tested each one with a variety of test sets. The real human data on each of those test sets was the same, but the non-human data was generated from each of the techniques in Section 3.4. We did this was to determine how effective each of the trained classifiers were against each of the adversarial methods. A server could then combine various versions of the trained classifiers if they appeared to be effective at detecting different sets of adversarial techniques.

5.1.2 Preventing Replay Attacks

To determine our ability to prevent malicious attacks, we deployed our trained classifier on the CAPTCHA server described in Section 4, and tested its ability to either detect a non-human user altogether, or prevent a non-human user from doing a replay attack using the technique we described in Section 4.4. In the Results section, we report the accuracy of this method on both detecting malicious attacks through the classifier or catching them through the replay detection technique.

	Test Accuracy	Test Set FP Rate	Test Set FN Rate	Steps Accuracy	Diffs Accuracy	Unif Accuracy	Gaussian Accuracy	HMM Accuracy
Steps Model	98.636%	0.4545%	0.9091%	99.2%	84%	79%	0%	3.6%
Diffs Model	97.273%	1.8181%	0.9091%	92.545%	97%	82%	0.73%	1.9%
Unif Model	99.545%	0%	0.455%	35.636%	30.182%	99%	0.364%	0.4%
Gaussian Model	46.818%	10%	43.1818%	22.909%	33.818%	8.364%	80.3%	82.9%
HMM Model	96.452%	3.226%	0.323%	68.182%	8.182%	18.91%	0.364%	93.8%
Mixed Model	79.545%	13.182%	7.273%	99.273%	99.455%	99.455%	8.546%	11.5%
Mixed + HMM	71.061%	28.636364	0.303%	92.364%	94.364%	92.181%	0.182%	39.6%

Figure 2. Comparative accuracy of different trained models on adversarial data types.

5.1.3 Human Activity

Next, as we noted earlier, while these accelerometer data can seem innocuous, we hypothesized that it can actually be used to easily determine the activity that a user is performing. Moreover, we believe it can be used to do this without any feature engineering - in other words, from the raw data emitted from the accelerometer, we can train a simple model that can predict what the user is doing with high accuracy.

Thus, for this classification task, we split the data that we collected ourselves, which along with accelerometer timesteps had a label that indicated the activity we were forming among 5 classes. The task was then to learn a classifier that could predict between the 5 classes given the accelerometer data. In order to make the code-writing efficient, we were able to use the same Recurrent Neural Network architecture do this classification task, except in the last layer, rather than projecting down to just 2 classes, we were able to project to 5, and make predictions using these.

Then, we took the data that we collected from our surveyed users and ran it through the classifier to see what they appeared to be doing when they submitted our survey.

5.2 Results and Evaluation of Success

5.2.1 Human vs. Not-Human

Figure 2 shows the results of training each of the seven models. The first column shows the accuracy on the test set during training. The second two columns show the false positive (FP) and false negative (FN) rate during training. Human examples are considered positive in our terminology, so a false positive is a robot labelled as human and a false negative is a human labelled as robot. The next five columns show the accuracy of the model against previously unseen data of each adversarial type. In the accuracy columns, the row with the highest accuracy is highlighted. We observe that all of the models perform incredibly poorly (worse than 40% accuracy) on at least two of the five types of adversarial data. Therefore in practice, it seems that the most successful model would likely be an ensemble of two of the models trained such that a user is only considered human if both models classify the user data as human.

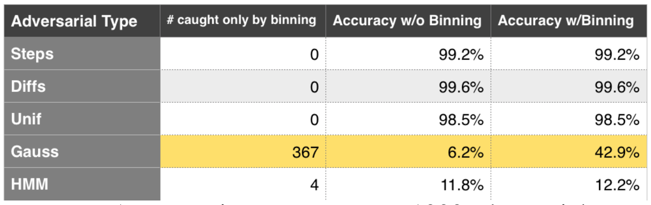
We propose two possible ensembles based on the client's preference for false positives versus false negatives. If a client is especially concerned about false positive (allowing robots through), then combining the Gauss model and the mixed model gives at least 80% accuracy in all categories of adversarial data. However, it also leads to a false negative rate greater than 43% which might be unacceptable. However, suppose that a website began collecting user motion data before clicking the "I am not a robot" button, in which case the client could verify a user as soon as they click the button or even before they do. In this case, a false negative would be no worse than presenting a traditional CAPTCHA to the users, and to the users who were verified successfully, they could bypass a traditional CAPTCHA.

If instead a client is not as concerned about letting through robots so long as most human users are allowed through, then combining the HMM model and the mixed + HMM model gives accuracy of at least 93% for all adversarial data types except for the Gaussian noise data (although as we discuss below, our replay-prevention techniques in part guard against Gaussian noise). However, this model has a comparatively high false positive rate (at least 26%).

Overall, we observe that models typically perform poorly against types of adversarial data different than those they were trained on. To some extent, it may be possible to combat this by using more training data (both positive and negative) than we were able to collect and train on, or by trying to exhaustively guess all the types of non-human data an adversary may send. However, a fundamental limitation of our approach is that it creates a sort of "arms race" between the creator of the CAPTCHA server and an adversary. This is a typical problem faced by the creators of CAPTCHA-like systems.

5.2.2 Preventing Replay Attacks

We enabled the binning systems described in section 4.4 with 500 bins. This number was chosen based on evaluating how many legitimate human inputs were blocked as replays due to this scheme. Choosing 500 bins caused 77 out of 548 (approx 14%) to be blocked. Figure 3 summarizes the results of using the binning scheme to try to detect adversarial replays on a survey deployed with the mixed model described in section 5.1.1. The primary attack we wish to prevent using the binning scheme is taking legitimate human data and adding a small amount of zero-centered noise to the data. This threat is well-represented by the 0-centered Gaussian noise data and we observe that the binning system improves accuracy on Gaussian noise data by over five times. It is not effective at preventing other sorts of adversarial data. The binning system in combination with either of the two ensemble schemes proposed above helps to provide more robust coverage.



Adversarial Type	# caught only by binning	Accuracy w/o Binning	Accuracy w/Binning
Steps	0	99.2%	99.2%
Diffs	0	99.6%	99.6%
Unif	0	98.5%	98.5%
Gauss	367	6.2%	42.9%
HMM	4	11.8%	12.2%

Figure 3. Accuracy improvements on 1000 adversarial samples due to our binning technique.

Note that if the server maintains a complete log of all of the binning hashes it receives, it will always prevent exact replay attacks. However, we observed above that the system also leads to denying a significant number of legitimate users as replay attackers. Therefore, there is a trade-off in implementation in choosing how many hashes to keep or how long to keep hashes, again depending on the user's tolerance for denying legitimate users as opposed to admitting replay attackers.

5.2.3 Human Activity

Next, we trained the second type of classifier on the data that had activities labeled. Then, we tested the accuracy of this model on a held-out test set, and got 76% accuracy on that data set. This means this classifier was able to predict the vast majority of the test data correctly, despite being trained on nothing more than the raw accelerometer data that we did not filter or engineer in any way.

Furthermore, we looked at the distributions of errors, and found that approximately 3/4 of the errors occurred in cases where a walking sample was incorrectly classified as in a car or on the stairs. We attribute this to the fact that walking shares many accelerometer-measured properties with these other tasks. We are confident that having a much larger dataset (ours was around 200 samples), as a large technology company may have, would substantially improve the performance on this task.

Lastly, we ran this trained model on the data that we collected from on survey, which did not include any annotations, and we used that to generate a set of predictions about what our surveyed individuals were doing:

Task	Number Predicted
walking	43
sitting	86
table	19
stairs	6
car	59

Table 1. Predicted action of Surveyers

The larger than expected number of students in a car is consistent with our earlier statement that this classifier tended to misclassify walking samples as car samples. Note that a model trained with more data would likely perform much better at this task, as it would be able to distinguish between tasks with greater granularity.

6. Conclusions and Future Work

6.1 Comparison to current reCAPTCHA

In the Related Work section, we discussed how parts of the current reCAPTCHA model could be beat with 70% accuracy. Our results show that combining the Gauss Model and the Mixed Model has a less than 20% error rate at every attack vector we tried. However, it has a high false negative rate (that is, many legitimate users are classified as suspect). As we discussed earlier, this is not a substantial problem, as a false negative just means that another reCAPTCHA test must be used. Furthermore, this technique is much more usable than the current reCAPTCHA on mobile, which would simply default to asking the user the image-based prompt. The majority of users, based on our data, would be able to get approved without any additional image-based task. In fact, the web site could completely hide the "I'm not a robot" button and validate the agent invisibly, similar to what Google's upcoming version of reCAPTCHA purports to do.

6.2 Privacy Discussion

This should present some serious privacy implications. Note that all of this motion data can be collected without permission requests from our users. All of it is accessible through well-known JavaScript functions in both Google Chrome and iOS Safari. A server with more computing capacity and data collection capability could quickly and clandestinely collect hundreds of thousands of samples (potentially labeled by cross-referencing user activities with their usage of other applications like Google Maps Navigation) which can be used to train highly-accuracy classifiers that determine what users are doing. The fact that a remote user can determine what a user is doing with high probability without any permissions request poses privacy concerns, and we believe it would be prudent for Google to clarify whether they will use such data sources as a component in their upcoming invisible reCAPTCHA.

Broadly speaking, any company that is claiming to create a motion-based CAPTCHA system as we have described could be masquerading intentions to collect large amounts of motion data to get more information about user behavior. We are not claiming that Google wishes to do this, but because Google's profit model is based on understanding user activity and targeting advertisements accordingly, Google should clarify its intentions and capabilities when developing a system that could use accelerometer data.

6.3 Future Work

There are several avenues for future work that we and others can take on. For one, recurrent neural networks generally require large amounts of data to train effectively, and so getting a large dataset of motion data from accelerometers, along with annotated data about human actions would go a long way towards creating an effective system for classifying behaviors.

Furthermore, the nature of CAPTCHA systems is fundamentally adversarial, where cunning enemies develop increasingly unique ways to beat the CAPTCHA model, and the designers of the CAPTCHA have to update the models to account for those techniques. In line with that, we identified the Hidden Markov Model (HMM) as a simple, effective model at getting through the model we presented, and thus a deployed version of this model must be trained to be resistant to such generated data, perhaps by generating a very large dataset of HMM data.

Lastly, given the ease with which developers can incorporate reCAPTCHA into their websites, and the fact that Google is moving to make it invisible from users altogether, there should be significant analysis of the privacy implications of any such technology, particularly if users are not asked for permission to use accelerometer data, as is currently the case. We have shown that, at the very least, even a small amount of training data can train a model with sufficient accuracy to guess, with high probability, what broad class of actions a user is doing. With thousands or millions (or billions) of training samples, it is likely that an organization can extend this capability to gain much more nuanced information, which can present serious risks that should be better understood and potentially mitigated.

References

[1] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford, “Captcha: Using hard ai problems for security,” in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 294–311, Springer, 2003.

[2] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, “Multi-digit number recognition from street view imagery using deep convolutional neural networks,” *arXiv preprint arXiv:1312.6082*, 2013.

[3] S. Sivakorn, I. Polakis, and A. D. Keromytis, “I am robot:(deep) learning to break semantic image captchas,” in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pp. 388–403, IEEE, 2016.

[4] A. Greenberg, “Google can now tell you’re not a robot with just one click.” `https://www.wired.com/2014/12/google-one-click-recaptcha`, 2014.

[5] J. H. J. S. Jeremy Elson, John (JD) Douceur, “Asirra: A captcha that exploits interest-aligned manual image categorization,” in *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*, Association for Computing Machinery, Inc., October 2007.

[6] M. Chew and J. D. Tygar, “Image recognition captchas,” in *International Conference on Information Security*, pp. 268–279, Springer, 2004.

[7] R. Gossweiler, M. Kamvar, and S. Baluja, “What’s up captcha?: a captcha based on image orientation,” in *Proceedings of the 18th international conference on World wide web*, pp. 841–850, ACM, 2009.

[8] P. Golle, “Machine learning attacks against the asirra captcha,” in *Proceedings of 15th ACM Conference on Computer and Communications Security (CCS)*, Association for Computing Machinery, Inc., October 2008.

[9] R. Gafni and I. Nagar, “The effect of captcha on user experience among users with and without learning disabilities,” in *11th Chais Conference for the Study of Innovation and Learning Technologies: Learning in the Technological Era*.

[10] C. A. Fidas, A. G. Voyiatzis, and N. M. Avouris, “On the necessity of user-friendly captcha,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2623–2626, ACM, 2011.

[11] V. Shet, “Are you a robot? introducing “no captcha recaptcha”.” `https://security.googleblog.com/2014/12/are-you-robot-introducing-no-captcha.html`.

[12] Google, “recaptcha: Tough on bots, easy on humans.” `https://www.youtube.com/watch?v=GeibaHfYW9o`.

[13] K. Kuspa and T. Pratkanis, “Classification of mobile device accelerometer data for unique activity identification,” 2013.

[14] M. D. Network, “Mdn: Devicemotionevent.” `https://developer.mozilla.org/en-US/docs/Web/API/DeviceMotionEvent`.

[15] C. Szegedy, W. Zaremba, I. Sutskever, J. Bruna, D. Erhan, I. Goodfellow, and R. Fergus, “Intriguing properties of neural networks,” *arXiv preprint arXiv:1312.6199*, 2013.