Generating Human-like Motion to Defeat Interaction-Based CAPTCHAs

Matthew Moore and Kristen R. Walcott

University of Colorado Colorado Springs, Colorado Springs, CO 80918, USA
mmoore7@uccs.edu, kwalcott@uccs.edu

Abstract. As more companies implement CAPTCHA systems to try to prevent automated attacks, CAPTCHA creators are increasingly using machine learning to try to filter out unwanted traffic. These systems are increasingly important in the development and maintenance of many web-based applications. As machine learning has evolved, so have the detection methods to block automated web traffic. As a result, some image-based CAPTCHAs are being replaced with systems that analyze mouse movements of the user to identify how likely it is that the user is human.

In this research, we develop and evaluate a 2-layer convolutional neural network driven framework that generates human-like motions. These types of movements are tracked by some CAPTCHA systems. We demonstrate that the framework's automatically generated movement paths can effectively and efficiently trick a classifier trained on features that are extracted from paths generated by humans. Using a 2-feature classifier as a CAPTCHA that was trained to recognize 91% of the human paths as valid human paths from our dataset, we are able to successfully bypass the CAPTCHA 89.25% of the time.

Keywords: Web Application Security, Machine Learning, interaction-based CAPTCHA

1 Introduction

Many websites use Completely Automated Public Turing tests to tell Computers and Humans Apart (CAPTCHAs) to prevent automated web traffic (bots). CAPTCHAs are designed to protect websites against bots posting fake reviews, stealing data from compromised accounts, posting phishing scams, manipulating contests/polls, and disrupting normal website functions. CAPTCHAs have many different forms. Some CAPTCHAs require the user to type the text that is spoken in an audio clip. However, the most ubiquitous CAPTCHAs are visual puzzles that the user must solve before logging into a website. The primary goal of CAPTCHAs is to be as easy as possible for humans to solve while blocking most automated requests.

According to the design goals, a CAPTCHA can be considered successful if it allows humans to solve a test successfully 90% of the time while allowing no

Fig. 1. reCAPTCHA V1 [15]

more than 1% of automated attacks to solve the test [14]. However, a Stanford study found that users were able to solve Google's text-based CAPTCHAs (reCAPTCHA V1 as seen in Figure 1) with 86% accuracy. The accuracy was much lower for Google audio CAPTCHAs, which had a solving accuracy of 35% [11]. Difficult CAPTCHAs frustrate users and harm the user experience. In some cases, bots may be able to solve CAPTCHAs more accurately and quickly than humans can.

To make CAPTCHAs easy to solve, modern CAPTCHA techniques involve using invisible tests that are interaction-based and monitor how the user moves the mouse, how they interact with webpages, and how frequently they visit those webpages. Interaction-based CAPTCHAs have the advantage that the user does not have to solve puzzles. This allows websites to only present traditional CAPTCHAs or block the user when the user is suspected of being a bot. To accomplish this, CAPTCHA providers collect several statistics about the user over a period of time and use machine learning to calculate a score that indicates the probability that the user is human. While the exact behavior of the machine learning algorithm are generally kept secret, security experts theorize that simple classifiers are used to detect abnormal users (bots).

The purpose of our research is to demonstrate the weakness of mouse-movement based CAPTCHAs and contribute to the understanding of mouse-based biometrics. We use machine learning to demonstrate that mouse movements can be generated from known real mouse movement behavior to bypass classifiers. To do this, we created a machine-learning based model to generate mouse movement paths. These paths are tested against a classifier that we created as a representative CAPTCHA system. This allowed us to accurately re-create and repeat tests. Using a 2-layer convolutional neural network (CNN) to generate the paths and a 2-layer Dense network to generate the timestamps, we created 10,000 datapoints that had the same destination goal as 10,000 human-generated paths. Out of the 10,000 paths generated by our AI network, 89.25% of the paths were detected as human by a 2-feature, one-class classifier that was trained using only human-generated paths.

In summary, the main contributions of this paper are:

1. Outline a framework to bypass an example captcha service (section 3)
2. Create an example neural network to generate output paths (section 4.1)
3. Detect paths as human or bot using a classifier as an example captcha service (section 4.2)
4. Demonstrate potential vulnerabilities in mouse-movement based captchas (Section 5.3)

2 Background

Most CAPTCHAs have used problems that are difficult to reliably program a computer to solve such as image recognition or language processing. Thus, successful CAPTCHAs are easy for humans to solve, but not un-solvable for computers. Computers may be able to solve them, but at a high cost-time price; they often require powerful computers, significant time, or large data-sets. This reduces the number of automated attacks that can be done against a target.

Early CAPTCHAs were simply text embedded into images. The text is usually distorted to make identifying text harder using optical character recognition. These text-based CAPTCHAs are relatively simple for algorithms to solve. Although these CAPTCHAs were reportedly the most common type, they have largely been replaced with better CAPTCHAs such as ReCAPTCHA which is now one of the most common, types [18]. Google, which owns reCAPTCHA, removed support for their text-based CAPTCHAs known as reCAPTCHA v1. Amazon still uses their own implementation of text-based CAPTCHAs.

Because of the weaknesses of text CAPTCHAs, image based CAPTCHAs are often used. The most common of these is the Google reCAPTCHA v2. With image CAPTCHAs, the user selects all the images that match what the CAPTCHA requires. These types of CAPTCHAs are more difficult to solve and require more resources, however they are quite solvable. One 2016 study was able to solve image CAPTCHAs with 41% accuracy and took only 20.4 seconds to solve them [30].

Audio CAPTCHAs are often presented along with image or text based CAPTCHAs. These CAPTCHAs are designed to present an audio-alternative to image based CAPTCHAs for visually impaired users. Audio CAPTCHAs, however, have long been a target for attackers. In 2009, researchers were able to solve Google's image reCAPTCHA V2 CAPTCHAs with 67% accuracy to obtain an exact match. In addition, the researchers found that even the general public could solve audio CAPTCHAs with 70% accuracy [33]. As audio recognition has improved, Google has implemented features to make audio CAPTCHAs more difficult to break, such as adding noise and creating more complicated audio phrases to solve. However, in a more recent paper, researchers were still successful at breaking audio CAPTCHAs. In that paper it was shown that, using publicly available audio recognition software, audio reCAPTCHAs were solved with greater than 83% accuracy [10]. The researchers found that other audio CAPTCHAs were much more difficult to solve using speech recognition.

Partly because of the reliability issues with audio CAPTCHAs, interaction based CAPTCHAs are becoming more popular. Interaction CAPTCHAs do not require the user to solve puzzles, but they instead ”invisibly” monitor the user input and behavior while the user is interacting with the webpage. This can be done by using javascript to monitor mouse movements. Thus, the user does not even know they are solving a CAPTCHA. In 2018, Google introduced reCAPTCHA

V3 which is an entirely interaction-based CAPTCHA system that is invisible to the user.

For security purposes, CAPTCHA companies do not fully disclose what information they track about users or how automated bots are detected using interaction based CAPTCHAs. However, some of these companies state that they do collect mouse movement, network traffic, cookie, and other data about the users. Security researchers speculate that these companies use this information as input into machine learning algorithms to determine if the user is human. Thus, if the user has suspicious mouse movements, the CAPTCHA service might subject the user to additional tests. Google's reCAPTCHA V2 image CAPTCHAs are reportedly protected by a checkbox that tracks mouse movements prior to clicking on the checkbox. If the mouse movements are determined to be suspicious, the user is presented with an image CAPTCHA.

3 CAPTCHA Automation Framework

Because user-interaction CAPTCHAs need to be very compliant to allow for a wide range of human interactions, this requirement could allow even poorly designed bots to mimic the behavior of a subset of users. Even though modern CAPTCHAs are thought to use mouse movement as one data point to determine if the user is human, this does not preclude attackers from automating mouse movements. Many bot-makers use extensive techniques to bypass CAPTCHAs because it can be very profitable. The goal of this project is to determine if motion based CAPTCHAs are a practical means for defeating online automation.

To accomplish this goal, we used a deep learning algorithm to generate mouse movement paths that mimic a set of human-like mouse movements. Although mouse movement is not the only aspect of user interaction that CAPTCHAs monitor, it is an aspect that is not often reproduced. Some CAPTCHAs list mouse movement analysis as a feature that is specifically monitored, and mouse interaction was determined to be an element of Google CAPTCHAs [1]. Testing this on real CAPTCHA services would be prohibitive because it would be difficult to isolate this single feature while testing. Thus, to test if the deep learning algorithm is successful, a machine learning classifier must be created instead to monitor important aspects of the path. A diagram that shows our method can be seen in Figure 2. Our method to create an algorithm for human-like mouse movement has several steps.

The first step is to manipulate a dataset into a usable form. Regardless of which dataset is used, the data must be in a consistent format. The dataset is a collection of mouse coordinates and timestamps that correspond to the path that a human took to complete a task such as clicking on a button. These paths are used as inputs to both the path generator and CAPTCHA emulator.

The next part of the diagram is the path generator, which is the piece that a bot would use to press buttons on a page to trick CAPTCHA services. It is responsible for creating the x,y coordinates and velocities for the mouse movements. For a given input destination coordinate, the path generator creates N number of coordinates and timestamps. The number of coordinates is determined by the length of the input paths. Because neural networks can be used to generate multiple outputs using learned parameters, they are ideal for this project. To allow more flexibility with training each of the models, we separated the path generator into two different neural network models. The first model creates the x,y coordinates for the path, and the second generates the timestamp data for each of the coordinates.

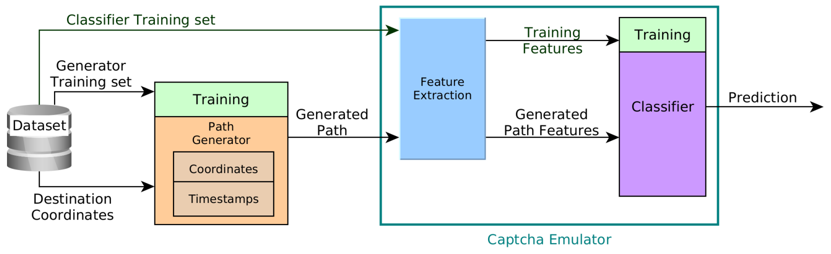


Fig. 2. Diagram of our process.

The final piece of the diagram is the CAPTCHA emulator which calculates features about the path so that the classifier can determine if the user is a human or not. In a real CAPTCHA, the features would be calculated client-side by obfuscated code before it is transmitted to the CAPTCHA provider's servers. Because commercial CAPTCHA service classifiers predict using multiple variables, our classifier behaves as a known quantity that we use to test our path generator. The classifier uses different calculated statistics (features) to predict whether an input path is human or a bot. The classifier is trained using a test dataset that is separate from the dataset used to train the neural network. We use the support vector machine (SVM) for our classifier because they are widely used in bot detection and can be configured in many different ways [35]. One-sided classifiers have the disadvantage that they can be less accurate than classifiers trained on a comprehensive dataset. However, we feel that this is a more realistic scenario for how real CAPTCHAs work. There are many possible ways that bots can move the mouse, such that training a classifier using them all would be impossible.

The classifier provides a determination about whether the features identify it as a human or non-human path. Each path falls into one of four categories:

– False negatives: human path classified as non-human
– True positives: human path classified as human
– False positives: non-human path classified as human

– True negatives: non-human path classified as non-human

4 Implementation

The two primary components of the framework are the path generator, which generates the human-like mouse movements, and the classifier, which serves as part of the CAPTCHA emulator.

4.1 Path Generator

To allow more flexibility with training each of the models, we separate the path generator into two different neural network models. The first model creates the x,y coordinates for the path, and the second model generates the timestamp data for each of the coordinates. For the path model, we use a 2-layer CNN that generates a 100 point path consisting of x and y coordinates for a pair of destination coordinates. The first CNN layer had a kernel size of 5 with linear activation and the second layer had a kernel size of 1 and linear activation. We chose a CNN because they are well-suited for spatial data such as coordinates and images. This model was trained using 10,000 paths (approximately half the human-generated paths datset) each of which contained 100 points. The second model creates an array of 100 timestamps that correspond to each of the points. This timestamp indicates the amount of time that has passed since the mouse first started moving. For this model, a Tensorflow dense neural network was used with 2 layers. Dense neural networks are deep learning networks similar to CNNs, but are more general purpose than CNNs. The first layer used 216 units with ReLU activation and the second layer used 100 units with ReLU activation. The timestamp model is trained on 10,000 different paths that each contained 100 timestamps. Both the coordinate and timestamp generator models used mean squared error for the loss function with an adam optimizer. Both models were trained for 100 epochs. After the paths are generated, they are scaled and normalized to make them more similar to natural paths. The x coordinates of the paths are normalized using formula 1.

$$x' = \frac{x - \text{min } x}{\text{max } x - \text{min } x} \tag{1}$$

After path normalization, the paths are rotated to ensure that the path end point was as close as possible to the input end point. This helps handle inconsistencies with the path generator to make sure that the the generated and user paths can be compared. Formula 2 shows the rotation matrix used to realign the paths.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2}$$

4.2 Classifier

We configured our classifier with an SVM and a radial basis function (RBF) kernel using Scikit-learn and Python. The RBF kernel classifies data in radial patterns to identify outliers. We used this kernel because it is simple to configure and can accurately detect clusters of similar data. We set ν, which controls the number of outliers/training errors in the training set, to 0.09 so that we have the desired 90% user-success (discussed in the metrics section) for the classifier while training. This parameter adjusts the size of the decision boundary.

For selected features, we selected two that we determined to be very important in determining how to classify a path. We used average mouse speed and distance over the minimum to the destination. The combination of these features helps the classifier more accurately differentiate humans from non-humans.

For average mouse speed, we measured the average distance moved from the previous point and the time delta from the previous timestamp to create the overall average speed for each of the paths. This feature is important because several tools, such as web testing tools, exist to move a mouse to a destination. These tools move the mouse instantaneously to the desired position so that buttons on web pages can be clicked on automatically. Thus, mouse paths with abnormally high velocities would be classified as non-human.

The distance over minimum feature calculates the difference between the total path length that the mouse has taken and the shortest path between the starting point and the path end point. An abnormally short path can indicate that a bot moved the cursor straight to the target, or an abnormally long path might indicate that random movements are done before the mouse was moved to the destination.

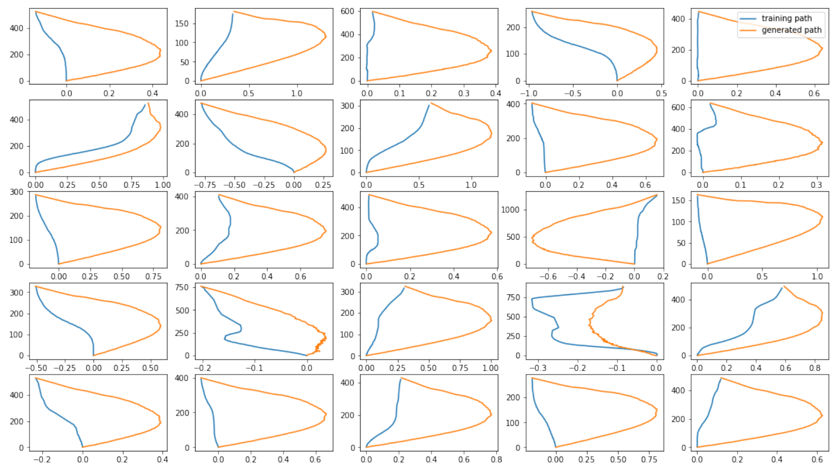


Fig. 3. A sample of the first 25 generated paths and natural paths.

5 Evaluation

We analyze the sample paths in Figure 3 to determine if our methods are successful. The generated paths do appear to be much longer and follow a more uniform shape. However, to properly analyze how our technique behaved, we must analyze the quantative results from the classifier.

We intend to show the following:

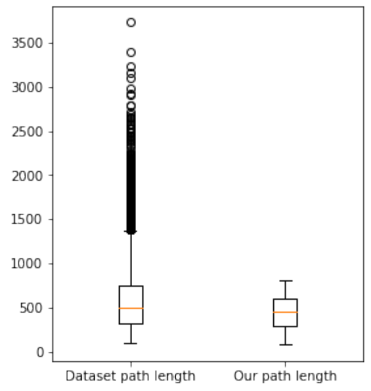
– The obtained dataset is a reasonable approximation of average human
– Our classifier follows the less than 10% false negative design goal for CAPTCHAs
– More than 10% of the paths from the path generator are classified as human by the classifier

5.1 Dataset

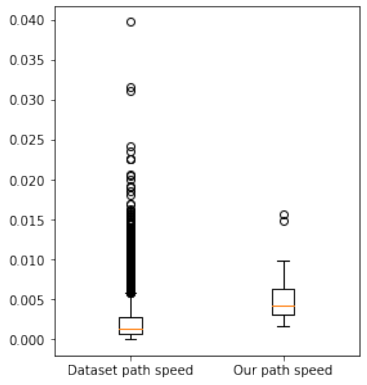
For our data, we use coordinates in standard Cartesian units (many programs generate coordinates with a reversed y axis). In addition, the coordinates all start at the origin and travel in the positive y direction to make comparing the paths easier. The x values of the dataset are scaled between -1 and 1 to improve the training rate for the path generator. The timestamps contain time data that starts at zero (or can be translated back to zero) and have millisecond accuracy to ensure that accurate time data is obtained for each point.

For our dataset, we use the data set that is included with the Github project *Natural Mouse Movements Neural Networks* [8]. This human-based data was used in [8] for a similar goal, to generate paths that are human-like, but the original project did not provide details about how the dataset was gathered from human subjects. The human mouse motion data provided by this project consists of 21,417 paths all consisting of 100 data points each. The paths have all been scaled between -1 and 1 for normalization among the paths. The data set also includes timestamps for each point in the path. Only human data was used in our study. To avoid influencing the results, the generator and classifier are trained with separate subsets of the dataset. The dataset is divided into two groups of approximately 10,000 paths each.

To demonstrate that our dataset of human-based mouse movement is representative of human user data outside of their study, we recorded 100 paths and timestamps to compare to the dataset. Our paths are collected from a single human subject using software that is running on a local machine (non-web based). Before recording is started, the mouse cursor is positioned in approximately the same point of the screen (bottom center). When the user is ready, a button is pressed which starts a mouse recording application named *cnee*. After starting the recording, the user moves the mouse cursor to a randomly selected icon at the top of the screen. When the user reaches the icon, they click a mouse button which stopped the recording software. Structuring our recording this way reduced the amount of post-processing necessary to create a uniform dataset. The mouse paths are recorded in the format $[(x_0, y_0, t_0), (x_1, y_1, t_1), ...(x_n, y_n, t_n)]$ where x_n is the x coordinate of the mouse movement, y_n is the y coordinate of the mouse movement, and t_n is the timestamp of the mouse movement.



(a) Path total length boxplot



(b) Path average speed boxplot

Fig. 4. Data validation between our collected paths and the dataset

These paths are then normalized so that they are in a similar format to the Github dataset. Overall, our recorded paths are very similar to our selected dataset. Our average path length was 446 (pixels) with a standard deviation of 190.146, and the dataset had an average path length of 584.42 with a standard deviation of 365.74. For the mouse movement speed, our paths had an average speed of 0.0049 (pixels/second) with a standard deviation of 0.0025 and the dataset had an average speed of 0.0021 with a standard deviation of 0.0023. Figure 4 shows a boxplot comparing our collected data to the dataset. Figure 4(a) shows that our paths are approximately the same average length with the dataset having many more outliers, which is expected for this kind of dataset. Figure 4(b) shows that our collected paths are, on average, slightly faster. This could be due to the way that we collected the paths. Because we collected our paths in rapid succession, the tendency was for the human to move the mouse faster than usual. Overall, however, the path speeds are reasonably close to the dataset, with our average speed falling within the 2 sigma range of the dataset average.

One of the other datasets that we considered for this project was the Balabit Mouse Challenge Data Set [21]. While this dataset did provide some details how it was collected, it did have some problems. This dataset, which was collected over remote desktop software, contains large jumps in the paths. In addition, there are many paths that have conflicting timestamp data. While some of these problems could be overcome, we decided to use the Natural Mouse Movements dataset because of the more consistent data.

5.2 Metrics

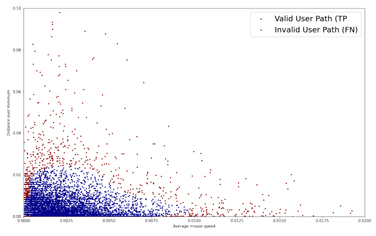
To measure how effective our method is in bypassing the classifier, we record the number of false positives and false negatives. We then measure the success using the number of false negatives (the number of times that the AI generated path is detected as a human). The number of false positives can also be used to determine the effectiveness of our bot-detector classifier. A classifier that detects too large of a percent (10%) of valid user inputs as non-human (false positive) does not meet the design criteria for CAPTCHAs and is invalid for our tests. A false positive detection rate of 10% would be a worst-case scenario for a classifier to achieve a 90% accuracy (the stated design goal for CAPTCHAs) for a user solve rate. The number of false positives and negatives is also used to calculate the F-score to show the overall performance of the classifier. Because CAPTCHAs are designed with a missed detection rate of 1 in 10,000, a false negative rating by the classifier of 1% or greater is considered a success [14].

5.3 Results

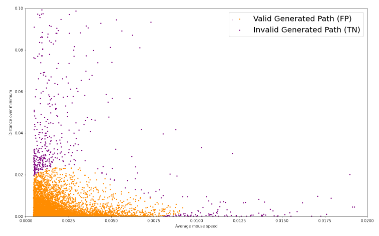
From Figure 3, we can see that our path generator tended to create parabolic paths based off of the training input. The paths are not perfectly smooth but had some ripple. For several of the paths, the endpoint did not match the designated end points. This was fixed using the matrix transformations discussed in Section 4.1 to rotate the paths to have the correct endpoint.

Out of the 21,417 paths, a total of 20,000 were used for training (1,417 were not used). The first 10,000 randomly selected paths (group 1) were used for training the path generator, and the second 10,000 randomly selected paths (group 2) were are for training the classifier. After training, the path generator is instructed to generate an additional 10,000 paths (group 3) using the starting and ending points of all the paths from group 1. By constructing the dataset this way, we obtain 10,000 human paths (group 1) and 10,000 generated paths (group 3) that can be easily compared because they have identical starting and destination points. In addition, these two groups do not bias the classifier results because they are not used to train the classifier.

To analyze how effective the path generator was at replicating the human paths, we analyze the results of the output of the classifier. Out of the 20,000 data points presented to the classifier (the 10,000 human paths from group 1, and the 10,000 generated paths from group 3), 18,023 paths are detected as human. As set by the design criteria, the 10,000 human-generated paths are detected as human greater than 90% of the time (90.98%) using our two-feature classifier. Figure 5(a) is a feature plot that shows which human paths are classified as human based off of their features (x-axis is the average mouse speed, y-axis is path distance over minimum). For our generated paths, the classifier classified 89.25% (8,925) of the paths as human. The feature plot of the generated data that is shown in Figure 5(b) shows that the path generator created many paths with features similar to the human-created paths (x-axis is the average mouse speed,

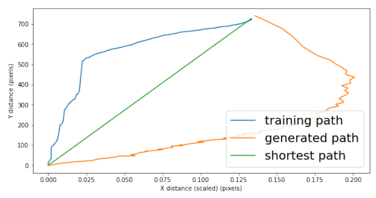


(a) Feature plot of the human paths

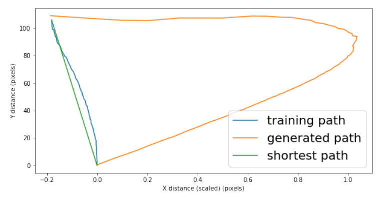


(b) Feature plot of the generated paths

Fig. 5. Velocity and distance extracted features.



(a) Generated path classified as human



(b) Generated path classified as non-human

Fig. 6. Comparison of generated and natural paths

y-axis is path distance over minimum). The human paths, however, were more evenly distributed along the x axis while the generator has more paths clustered near the origin This resulted in a high false positive rate for the classifier. The F-score (which is an indication of the performance of classifiers) for our model is 0.649 and the model accuracy is 50.87%.

Although the generated paths in Figure 3 look significantly different than the paths they are supposed to replicate, this is not necessarily an issue. The goal of the generator is not to create paths that look identical to one specific path, but rather to generate paths that are similar in shape, length, and speed to human paths. Figure 6 shows an example of two paths that look very similar, but are classified differently. The destination in Figure 6(a) is much further away from the origin than the path in Figure 6(b). As a result, the classifier expects the path to be much longer, and has more tolerance for those paths that are longer.

These results show that a simple path analysis system that is configured to allow 90% of human traffic could be defeated 89.25% of the time by bots. By using relatively simple machine learning techniques, bots can generate paths that have simple curves and varying path velocities. This lowers the classifiers F-score (the overall measure of performance) possible to 0.649. In total, using path length and mouse velocity as features, only 50.1% of the paths were able to be correctly identified.

The results can be summarized as follows:

———————Human Paths———————
True positives: 90.98%
False positives: 9.02%
———————Generated Paths———————
True negatives: 10.75%
False negatives: 89.25%
———————General Statistics———————
Total paths detected as human: 18023
Total paths detected as bot: 1977
F-score: 0.649324
Accuracy: 0.50865

6 Threats to Validity

There are several possible validity threats with this project. One threat is that our method may not accurately represent how CAPTCHA companies actually detect bots. Although several CAPTCHA companies list mouse movement as an attribute that is examined when determining if the user is a computer or human, the method of detection and techniques used is not described in detail in the existing research literature. Many of these companies intentionally keep these aspects of their software secret to make defeating CAPTCHAs harder. Thus, the goal of this project is to demonstrate that attackers can take a generic approach to replicating human behavior without any special knowledge of what parameters the CAPTCHA is monitoring.

Another possible threat is the size of dataset used for this project. Although the mouse movements are generated from human users, the dataset only contains the input from a relatively small sample set of users. Different users might have more widely varying patterns to their input. While this does to some extent modify the results that the deep learning will produce, the expected results should still be similar enough to a human to bypass classifiers.

While there may be other classifiers that are more accurate and efficient, this research is to show the feasibility of using common-place classifiers to defeat CAPTCHAs. Other more advanced learning algorithms will likely become more common in the future.

7 Related Work

Breaking image and text based CAPTCHAs with machine learning is a well-researched topic. A study from 2003 found that image recognition was able to identify text CAPTCHAs 93% of the time [25]. Furthermore, many more researchers have studied methods for breaking text-based CAPTCHAs [13,36,34,2]. In addition, image based CAPTCHAs have been reliably solved. One study was able to solve image reCAPTCHA's with an accuracy of 71% [30]. Because interaction based CAPTCHAs have only recently increased in popularity, less research exists in this field.

One related article is *Hacking Google reCAPTCHA v3 using Reinforcement Learning*. In this article, researchers used reinforcement learning to move a mouse cursor to the check box of a reCAPTHCA v2 [3]. Using a grid, the researchers would increase a "reward" score for the algorithm as the mouse was moved in a grid closer to the target. However, this project did not have the goal of tricking classifiers by replicating human input, but instead focused on simply reaching the goal location using reinforcement learning.

Another area of related research is from the article *Intrusion Detection Using Mouse Dynamics* [6]. In this article, the researchers use the Balabit Mouse Challenge Data Set [21] mouse movement data to identify different users to detect when accounts are compromised. They do some useful analysis of the Balabit mouse data [7].

In *An Insider Threat Detection Approach Based on Mouse Dynamics and Deep Learning*, researchers used the Balabit Mouse Data Set to analyze human mouse movement to detect unusual behavior in users. The researchers converted all of the mouse movement paths and actions to images and analyzed the behaviors of each of the different users using a convolutional neural network. The behaviors of each of the users was compiled in order to analyze different aspects that uniquely identified a user [22]. This project showed that mouse movements can be used as a unique identifier for users.

Much research also exists from the other perspective of the detection of bots attacking CAPTCHA systems. Some of these works simply attempt to recognize web bots from human visitors ([4], [5], [12], [16], [20], [29]). Others consider the abilities and complexities of the bots and their detection evasiveness ([19], [9], [28], [37], [24]). Bot detection has also been performed using web logs ([4], [16], [17], [27], [29], [31], [32]) and mouse movement tracking ([26],and [24]. Many of these techniques are machine learning based for the purpose of detection. In this work, we demonstrate that machine learning can be used to defeat mouse based interaction models. Humanlike behavior including the performance of mouse movements has been shown to be difficult to detect ([24] and [23]).

8 Conclusion

Our work contributes to the understanding of online security and helps CAPTCHA companies to better understand weaknesses with existing systems. We use common machine learning algorithms to generate paths that are difficult for a classifier to identify. This research helps further knowledge about potential CAPTCHA vulnerabilities and provides motivation for companies to develop methods to counter attacks, such as the attack presented in this paper.

Provided that attackers can obtain a database of several thousand users' mouse movements, they could use mouse-movement as an attack vector to increase their chances of successfully bypassing existing CAPTCHA services without the need for human interaction or CAPTCHA solving services. Although mouse movement might function as a possible variable that can be examined to filter automated traffic, it should not be used as the sole identifying characteristic. Through our framework, we demonstrate the ease with which attackers can bypass CAPTCHAs that only use mouse movement. Our 89.25% success rate of tricking a classifier demonstrates that using mouse movement as a bot-prevention technique should be used with caution.

For future work, we would like to include more features for the classifier. More features would allow the classifier to potentially better identify paths. Some features we could consider adding are path curvature, number of times the path changes direction, or possibly starting/ending velocity. Another area of future work would be to improve the classifier with different kernels. With a polynomial kernel, we could obtain a more strict decision boundary for the classifier. This would make it more difficult for bots to bypass the classifier. The path generator model could also be advanced by incorporating more layers with larger filter sizes. In addition, layers can be added to introduce randomness into the path to make the output of the generator less regular.

Humanlike movements could also be combined with other metrics that could be used in CAPTCHA bot detection technology. An extended framework could take multiple data sources into consideration. Also, this framework could be evaluated based on interation-based bot detection tools.

References

1. Recaptchareverser (2014). URL https://github.com/neuroradiology/InsideReCaptcha
2. Breaking text-based captchas with variable word and character orientation. Pattern Recognition **48**(4), 1101 – 1112 (2015). DOI https://doi.org/10.1016/j.patcog.2014.09.006
3. Akrout, I., Feriani, A., Akrout, M.: Hacking google recaptcha v3 using reinforcement learning (2019)
4. Alam, S., Dobbie, G., Koh, Y.S., Riddle, P.: Web bots detection using particle swarm optimization based clustering. In: 2014 IEEE congress on evolutionary computation (CEC), pp. 2955–2962. IEEE (2014)
5. AlNoamany, Y.A., Weigle, M.C., Nelson, M.L.: Access patterns for robots and humans in web archives. In: Proceedings of the 13th ACM/IEEE-CS joint conference on Digital libraries, pp. 339–348 (2013)
6. Antal, M.: Intrusion detection using mouse dynamics. IET Biometrics **8**, 285–294(9) (2019)
7. Antal, M., Egyed-Zsigmond, E.: Mouse Dynamics – Measurements on the Balabit Data Set (2019)
8. Artmann, D.: Natural Mouse Movements (2019). https://github.com/DaiCapra/Natural-Mouse-Movements-Neural-Networks

9. Bai, Q., Xiong, G., Zhao, Y., He, L.: Analysis and detection of bogus behavior in web crawler measurement. Procedia Computer Science **31**, 1084–1091 (2014)
10. Bursztein, E., Beauxis, R., Paskov, H., Perito, D., Fabry, C., Mitchell, J.: The failure of noise-based non-continuous audio captchas. In: 2011 IEEE Symposium on Security and Privacy, pp. 19–31 (2011)
11. Bursztein, E., Bethard, S., Fabry, C., Mitchell, J.C., Jurafsky, D.: How good are humans at solving captchas? a large scale evaluation. In: 2010 IEEE symposium on security and privacy, pp. 399–413. IEEE (2010)
12. Cabri, A., Suchacka, G., Rovetta, S., Masulli, F.: Online web bot detection using a sequential classification approach. In: 2018 IEEE 20th International Conference on High Performance Computing and Communications; IEEE 16th International Conference on Smart City; IEEE 4th International Conference on Data Science and Systems (HPCC/SmartCity/DSS), pp. 1536–1540. IEEE (2018)
13. Chandavale, A.A., Sapkal, A.M., Jalnekar, R.M.: Algorithm to break visual captcha. In: 2009 Second International Conference on Emerging Trends in Engineering Technology, pp. 258–262 (2009)
14. Chellapilla, K., Larson, K., Simard, P.Y., Czerwinski, M.: Building segmentation based human-friendly human interaction proofs (hips). In: Human Interactive Proofs, pp. 1–26. Springer Berlin Heidelberg, Berlin, Heidelberg (2005)
15. Chen, J., Luo, X., Guo, Y., Zhang, Y., Gong, D.: A survey on breaking technique of text-based captcha. Security and Communication Networks **2017**, 6898,617 (2017). DOI 10.1155/2017/6898617
16. Chu, Z., Gianvecchio, S., Wang, H.: Bot or human? a behavior-based online bot detection system. In: From Database to Cyber Security, pp. 432–449. Springer (2018)
17. Dewa, Z., Maglaras, L.A.: Data mining and intrusion detection systems. International Journal of Advanced Computer Science and Applications **7**(1), 62–71 (2016)
18. Dionysiou, A., Athanasopoulos, E.: Sok: Machine vs. machine - a systematic classification of automated machine learning-based captcha solvers. Computers Security **97**, 101,947 (2020)
19. Doran, D., Gokhale, S.S.: A classification framework for web robots. Journal of the American Society for Information Science and Technology **63**(12), 2549–2554 (2012)
20. Doran, D., Gokhale, S.S.: An integrated method for real time and offline web robot detection. Expert Systems **33**(6), 592–606 (2016)
21. Fülöp, A., Kovács, L., Kurics T., W.P.E.: Balabit Mouse Dynamics Challenge data set (2016). https://github.com/balabit/Mouse-Dynamics-Challenge
22. Hu, T., Niu, W., Zhang, X., Liu, X., Lu, J., Liu, Y.: An insider threat detection approach based on mouse dynamics and deep learning. Security and Communication Networks **2019** (2019). DOI 10.1155/2019/3898951
23. Iliou, C., Kostoulas, T., Tsikrika, T., Katos, V., Vrochidis, S., Kompatsiaris, I.: Detection of advanced web bots by combining web logs with mouse behavioural biometrics. Digital Threats: Research and Practice **2**(3) (2021). DOI 10.1145/3447815. URL https://doi.org/10.1145/3447815
24. Iliou, C., Kostoulas, T., Tsikrika, T., Katos, V., Vrochidis, S., Kompatsiaris, Y.: Towards a framework for detecting advanced web bots. In: Proceedings of the 14th International Conference on Availability, Reliability and Security, pp. 1–10 (2019)
25. Mori, G., Malik, J.: Recognizing objects in adversarial clutter: breaking a visual captcha. In: 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2003. Proceedings., vol. 1, pp. I–I (2003). DOI 10.1109/CVPR.2003.1211347

26. Networks, D.: BAD BOT REPORT: The Bot Arms Race Continues (2019). https://resources.distilnetworks.com/white-paper-reports/bad-bot-report-2019
27. Rovetta, S., Cabri, A., Masulli, F., Suchacka, G.: Bot or not? a case study on bot recognition from web session logs. In: Italian Workshop on Neural Nets, pp. 197–206. Springer (2017)
28. Seyyar, M.B., Çatak, F.Ö., Gül, E.: Detection of attack-targeted scans from the apache http server access logs. Applied computing and informatics **14**(1), 28–36 (2018)
29. Sisodia, D.S., Verma, S., Vyas, O.P., et al.: Agglomerative approach for identification and elimination of web robots from web server logs to extract knowledge about actual visitors. Journal of Data Analysis and Information Processing **3**(01), 1 (2015)
30. Sivakorn, S., Polakis, J., Keromytis, A.D.: I'm not a human: Breaking the google recaptcha (2016)
31. Stevanovic, D., An, A., Vlajic, N.: Feature evaluation for web crawler detection with data mining techniques. Expert Systems with Applications **39**(10), 8707–8717 (2012)
32. Stevanovic, D., Vlajic, N., An, A.: Detection of malicious and non-malicious website visitors using unsupervised neural network learning. Applied Soft Computing **13**(1), 698–708 (2013)
33. Tam, J., Simsa, J., Hyde, S., Ahn, L.V.: Breaking audio captchas. In: Advances in Neural Information Processing Systems, pp. 1625–1632 (2009)
34. Tang, M., Gao, H., Zhang, Y., Liu, Y., Zhang, P., Wang, P.: Research on deep learning techniques in breaking text-based captchas and designing image-based captcha. IEEE Transactions on Information Forensics and Security **13**(10), 2522–2537 (2018). DOI 10.1109/TIFS.2018.2821096
35. Wang, P.: [tensorflow] ch4: Support vector machines (2018)
36. Yan, J., El Ahmad, A.S.: Breaking visual captchas with naive pattern recognition algorithms. In: Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007), pp. 279–291 (2007)
37. Zabihimayvan, M., Sadeghi, R., Rude, H.N., Doran, D.: A soft computing approach for benign and malicious web robot detection. Expert Systems with Applications **87**, 129–140 (2017)