Midterm Report: *I'm Not a Robot: CAPTCHA Design, Security & Attack*

Asmi Kawatkar, Kenan Cackovic, Kelly Couvrette, Jon Paino, Lucy Zises, Bianca Mittu

[1]University of California, Los Angeles (UCLA)

ABSTRACT This project delves into the development, vulnerability analysis and security enhancement of CAPTCHA systems, which play a crucial role in preventing automated bot attacks on web platforms. As CAPTCHA mechanisms are increasingly challenged by advancements in AI, machine learning and automated attack scripts, there is a pressing need to evaluate and innovate these systems. The project includes a literature review covering CAPTCHA evolution, from traditional text-based systems to complex image-recognition CAPTCHAs. Notably, the project explores the vulnerability of two types of CAPTCHAs - a slider-based image CAPTCHA and a text-based input CAPTCHA - by implementing targeted attack strategies. These include brute-force, OCR attacks and direct manipulation of CAPTCHA parameters through browser scripts. The slider-based CAPTCHA, for isntance, was shown to be easily bypassed due to predictable angles and lack of server-side validation. Similarly, the text-based CAPTCHA, using a static character set, presented vulnerabilities exploitable by simple OCR tools.

The project's key takeaways include the need for CAPTCHA systems to incorporate more dynamic elements, such as randomized starting conditions, server-side validation, and rate-limiting. An additional insight underscores the importance of balancing accessibility with security: while complex CAPTCHAs may improve bot resistance, they may also hinder user accessibility, especially for differently abled individuals. These findings underscore the need for continued innovation in CAPTCHA design to not only counteract evolving AI technologies, but also to balance user experience with security. This multi-phase approach integrate continuous literature review and practical testing & implementation ensures an up-to-date understanding of CAPTCHA resilience and potential improvements in modern security landscapes. *View source code here.*

INDEX TERMS Automated Attacks, Bot Detection, CAPTCHA, Cybersecurity, Web Security

I. INTRODUCTION

CAPTCHA systems are critical for maintaining web security by distinguishing between human users and automated bots. This functionality protects against malicious activities such as data scraping, spam and unauthorized access to sensitive information. However, as artificial intelligence advances, traditional CAPTCHA mechanisms face growing vulnerabilities, necessitating continuous evaluation and improvement.

II. PROJECT GOALS & TIMELINES

The main goal of this project is to bolster our understanding of CAPTCHA technology development, while deploying, attacking and evaluating the robustness of our own rudimentary CAPTCHA system. We seek to answer the following research questions:

- **Vulnerability Mapping:** What are the most common methods used to bypass CAPTCHA, and how effective are they against modern CAPTCHA versions?
- **AI Resilience:** How well do CAPTCHAs hold up against deep learning models?
- **Cost vs. Security:** What trade-offs exist between maintaining high security and the computational resources needed to develop and maintain complex CAPTCHA systems?

Regarding the timeline, we have planned the following:

- **Week 3-4**: Preliminary Literary Review/Build Background knowledge about the history/latest in the field of CAPTCHA security.
- **Week 5**: Implement a functional login page with CAPTCHA options (currently focusing on basic image & text based options).
- **Week 6**: Implement early versions of the attack/explore vulnerabilities in the existing login system.
- **Week 7**: Midterm Presentation
- **Week 8**: Implement more attacks & suggest defenses to existing attacks
- **Week 9**: Complete ongoing literature review, final report and final presentation slides.

III. SETUP

This project incorporates a simple yet effective login and registration system designed to be compatible with multiple CAPTCHA mechanisms, aiming to secure user interactions and protect against automated bot attacks. This sections outlines the core components of the login system, the authentication process and the existing CAPTCHA setup used to verify legitimate users.

Login and Registration System: The login/registration system is designed to manage user access through a basic login and registration flow. Users can register for an account by providing essential information such as a username and password as shown in Figure 1.

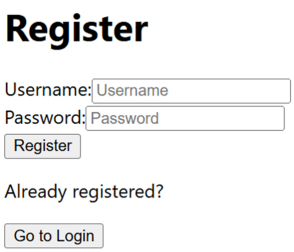

FIGURE 1. **Screenshot of login page**

Once registered, users can login by entering their username and password. The system authenticates these credentials by matching the entered information against stored values. If the credentials are correct, the user moves into the CAPTCHA stage.

FIGURE 2. **Screenshot of registration page**

To prevent automated login attempts, the system is integrated with a CAPTCHA mechanism that is presented to users during the login process.

CAPTCHA System: In the current implementation, there are 2 types of CAPTCHAs that the user might encounter:

- *Image Based*: The user is asked to drag a slider to rotate an image until the desired orientation is achieved (as per the instructions) and then click submit. The system then authenticates that the image is indeed pointing in the required direction, and verifies the login attempt accordingly.

FIGURE 3. **Image Based CAPTCHA: Rotate to match specified direction**

- *Text Based:* The user is asked to type in the shown string (case sensitive) and then click submit. The system then authenticates that the string input by the user matches the string displayed in the image, and verifies the login attempt accordingly.

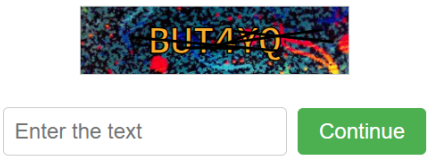

FIGURE 4. **Text Based CAPTCHA: Type in a matching string**

IV. THREAT MODEL

The threat model outlines a potential attack where an automated script is used to bypass a CAPTCHA verification system by extracting the CAPTCHA text using web scraping and image processing techniques. This attack targets web applications that use CAPTCHAs as a defense mechanism against bots and automated access.

A. Threat Actors

- **Primary Threat Actor:** Malicious developers or cybercriminals aiming to bypass CAPTCHA systems to perform unauthorized actions such as scraping data, spamming or brute-forcing.
- **Secondary Threat Actor:** Automated bots used by competitiors for reconnaissance or scraping purposes

B. Attack Vector

- **Entry Point:** The CAPTCHA component of a web page.
- **Method:** An attacker uses a script (e.g written in Node.js with Puppeteer) to automate the process of visiting the page, capturing the CAPTCHA image, and using optical character recognition (OCR) libraries to extract and interpret the text.

C. Assets at Risk

- **Protected Resources:** Access to pages or actions restricted behind CAPTCHA verification

- **User Data:** If the CAPTCHA is protecting forms or user accounts, unauthorized access could lead to data exfiltration.
- **Application Reputation:** Compromised CAPTCHA mechanisms may damage trust in the application's security.

D. Potential Impacts

- **Unauthorized Access**: Attackers may gain access to parts of the application/services meant to be safe-guarded by CAPTCHA
- **Automated Spam and Abuse:** Web apps can be vulnerable to spam, account creation abuse, or denial-of-service (DoS) through automated submissions.
- **Credential Stuffing**: CAPTCHAs often act as rate-limiting steps for login attempts. Bypassing them could expose the app to credential stuffing attacks[2].

E. Attack Steps

1) **Script Preparation**: Create a web scraping script to navigate the target webpage and capture the CAPTCHA component.
2) **Image Capture/Processing:** Use the script to take a screenshot of the CAPTCHA, and employ OCR libraries (e.g Tesseract.js) to process the image and extract the text (in the case of text based CAPTCHA).
3) **Submission:** Submit the extracted CAPTCHA text to bypass the verification step, and proceed to the protected part of the web application.

F. Assumptions & Limitations

1) **White Box Attack:** We assume for simplicity that the attacker is aware of the internal construction of the system, and can craft an attack based on this knowledge of how the CAPTCHA works (although in real world applications this might not be the case).
2) **Accessibility:** We assume that the user does not have visual impairments since these CAPTCHAs rely primarily on visual cues. However, we recognize that this is a strong assumption for the real world, and there is a need for alternative challenges like audio or CAPTCHA-solving assistance, which would be beneficial for a wider audience [5].
3) **Lack of Advanced Obfuscation**: The CAPTCHAs that we were able to implement up to this point lack advanced warping techniques (e.g pixel manipulation/character overlap).
4) **No Server-Side CAPTCHA Validation**: The CAPTCHA is entirely implemented on the frontend for this project, which may make it more vulnerable. If a bot can bypass the frontend JavaScript, it could submit the CAPTCHA answers directly by API calls, bypassing the need to solve the CAPTCHA entirely.
5) **No Logging or Tracking of Failed Attempts**: At this stage, our implementation does not track or log failed CAPTCHA attempts or identify patterns of suspicious behavior. Without this, bots could continue trying to bypass the CAPTCHA without triggering alerts for unusual activity.

V. CAPTCHA #1: Rotate an Image

A. Vulnerability Analysis

This implementation asks a user to rotate a certain image by a fixed angle and always presents the image from the same starting position. Due to its predictable nature, this CAPTCHA is vulnerable to automation as demonstrated in the attack described in the following section.

B. Attack

Given the predictable nature of the CAPTCHA, a simple attack can be deployed by executing the following lines of code in the developer console after entering login credentials:

```
document.querySelector('input[type
    ="range"]').value = 90;
document.querySelector('button').
    click();
```

Note that the second line must be executed twice.

The `querySelector` function allows you to access the input element of type "range", which is used to control the rotation of the image in the CAPTCHA challenge. This is a typical slider input element, where the value corresponds to the angle of rotation. Once you have access to the slider element, you set its value directly to 90 (since you know that the CAPTCHA will always require a 90 degree rotation). After setting the slider value, the attack proceeds by clicking the submit button (the second line). The first time this is executed, it fails because the internal code requires the degree of rotation to be updated to 90 degrees. However, in this implementation of the CAPTCHA, the 'submit' button also checks and updates the degree of rotation of the image/slider. So after the first execution of the automated submit, the degree of rotation is set to 90 and thus the second execution of submit passes the CAPTCHA test (shown in Figure 5).

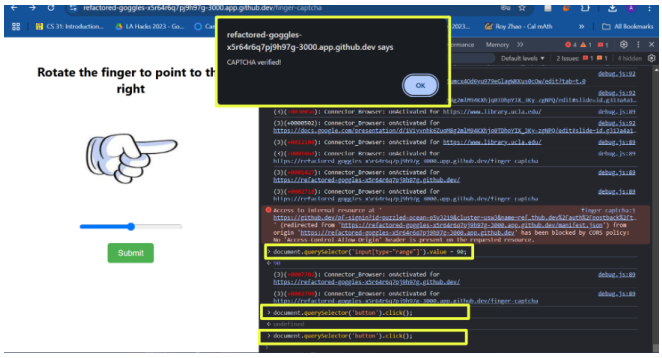

FIGURE 5. Illustrating the execution & success of attack on image rotation CAPTCHA.

Why it works:

1) **Fixed Angle**: The slider value represents a fixed angle, and if the CAPTCHA is always expecting a value 90, it becomes trivial for a script to automate the process by setting the slider directly to that value.
2) **No Dynamic Behavior**: The system doesn't randomize the required angle or the starting position of the image, meaning the bot doesn't need to perform any calculations or image processing. It simply needs to know the value that the slider should be set to, which can easily be hard-coded in the script.
3) **Direct Interaction**: By using `querySelector` to directly manipulate the Document Object Model (DOM), the script could bypass the need for real user input altogether. It simulates the exact behavior expected by the CAPTCHA system without any visual cues or user interaction.

C. Proposed Defenses

To prevent this kind of attack, the CAPTCHA system needs to have a higher level of complexity. Some potential countermeasures include:

1) **Randomized Angles:** For each CAPTCHA, generate a random angle (e.g., between 10° and 360°) and require the user to rotate the image to that angle. This randomness ensures the attacker cannot predict the correct angle to input.
2) **Slider Range Randomization:** Randomize the range of the slider itself for each challenge (e.g., the minimum and maximum values). This forces the script to handle the varying slider range instead of relying on a fixed value.
3) **Randomized Starting State:** Rotate the image by a random amount before presenting it to the user. This way, even if the bot sets the slider to a fixed value, the resulting rotation will not be correct because the initial state of the image is unknown.
4) **Timeout after X failed attempts:** If the slider is moved too quickly or if multiple actions are performed in a very short amount of time (e.g., clicking and manipulating the slider almost simultaneously), it could signal an automated attempt. Adding a timeout/rate limitation to the system would help to prevent automated CAPTCHA bypass. In fact, according to a 2024 study, replacing CAPTCHAs with rate limiters has shown have promising impacts in improving security and accessibility. [4]

D. Implemented Defenses:

Changing the logic of `handleSubmit()` as shown in the code snippet in Figure 6 causes the proposed attack to become ineffective. This is because while the slider is visually updated to the halfway point, the property of 'rotation' that is checked as a condition by `handleSubmit()` is not updated, causing the attack to fail.

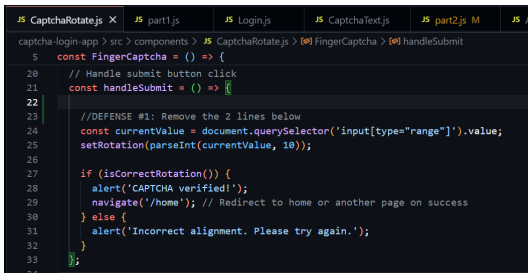

FIGURE 6. Defense for the Image Rotation Attack

VI. CAPTCHA #2: Typing a String

A. Vulnerability Analysis

One notable concern about this implementation is the predictability of the CAPTCHA text. Although the text is randomly generated, it uses a limited character set (capital alphabets & numbers) and a fixed length of 6 characters, reducing entropy and making it susceptible to brute-force or machine-learning-based prediction methods. The use of a static background image (`busyPattern.png`) could also assist attackers by providing a consistent visual template, making it easier for automated scripts to identify and isolate the CAPTCHA text.

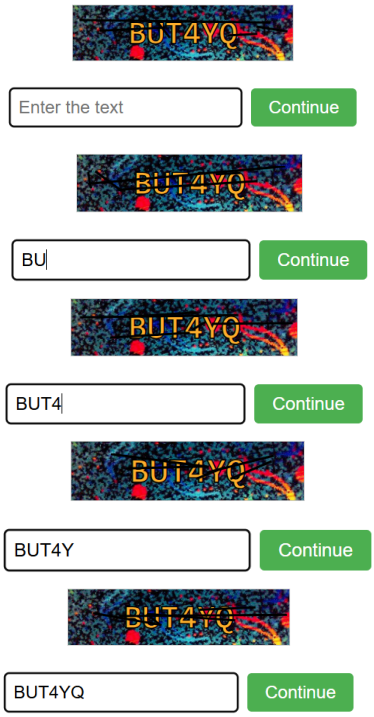

FIGURE 7. Randomized line overlay with every character typed

Additionally, the font and styling of the CAPTCHA text, including the consistent use of WebkitTextStroke, could simplify text extraction for image processing tools like

Optical Character Recognition (OCR). The SVG overlay with random lines (that change each time a character is typed - Figure 7) might add some visual noise, but this could be inadequate for preventing automated bots, as such lines can often be filtered out. Furthermore, the CAPTCHA lacks sophisticated visual challenges such as text rotation, warping or distortion, which are commonly used to make text harder for automated systems to recognize while still being legible to human users.

The handling of user input verification is conducted on the client side (`handleSubmit` function), revealing the logic used for checking the CAPTCHA. This client-side approach exposes the code to potential reverse engineering and could allow attackers to bypass the challenge by modifying or automating the verification process in their browser. Additionally, the absence of rate-limiting mechanisms or IP-based checks means that repeated failed attempts may go undetected, leaving the system vulnerable to brute-force attacks.

B. Attack

We are currently working on implementing this. The goal is to implement one of the following attacks:

- *Brute Force Attack*: Since the text CAPTCHA will always have length 6, there are $(26+10)^6 = 2176782336$ combinations of characters that an attacker might guess since this CAPTCHA does not involve rate limitation.
- *OCR Attack:* Attacker can use a tool like Tesseract.js to analyze the CAPTCHA image, extracting the CAPTCHA text automatically by reading visible characters from the background image, bypassing the need to manually input the characters. Since the CAPTCHA text is relatively simple and not distorted in any way, it would be relatively simple for an attacker to implement this.
- *Exploiting Lack of Server-Side Validation:* Attacker could bypass validation entirely by submitting the form without interacting with the CAPTCHA, manipulating requests directly through developer tools or an HTTP request tool.

C. Proposed Defenses

Depending on the attack we are able to implement, some potential defenses for this text based CAPTCHA system would be:

- *Increase CAPTCHA Complexity*: Introduce more complex background noise, wavy/rotated text and random distortions to the image. This would make it more difficult for OCR tools to interpret the image.
- *Use Time-Based Expiry*: Implement a short expiry time for CAPTCHA tokens (e.g 30-60s). This prevents attackers from capturing valid CAPTCHA tokens and using them in a future attack.
- *CAPTCHA Refresh*: For each attempt, refresh the CAPTCHA after a specific amount of time, ensuring that attackers cannot continuously solve it without user interaction.

VII. Related Work

Related work in the field of CAPTCHA systems has largely centered on balancing ease of use for human users with resistance to automated attacks.

A. CAPTCHA History & Examples

The CAPTCHA was originally created at Carnegie Mellon University in 2000[1].

Professors Manuel Blum, Luis A. von Ahn and John Langford from Carnegie-Mellon Universitiy identified following key properties of an ideal CAPTCHA system [3]:

- Quick and easy for humans to solve
- Low false negative rate (minimizing rejection of legitimate users)
- Low false positive rate (avoiding acceptance of bots)
- Resistant to automated attacks despite increasing sophistication of technology.

Early CAPTCHA systems were primarily text-based, where users were asked to decipher distorted characters. However, these text-based systems were vulnerable to attacks through optical character recognition (OCR) technology, which could be used to break down the distorted characters and solve the CAPTCHA automatically. As a result, image-based CAPTCHAs became a proper alternative.

These involve asking users to identify objects or patterns within images, a task that is generally easier for humans but more difficult for automated systems. One early example of this approach is labeled image CAPTCHAs, where users are asked to select images that contain a specific object, such as bicycles or traffic lights (Figure 8). First proposed in 2004, this orm of CAPTCHA was designed to exploit human ability to recognize complex objects and scenes, which remains a challenge for many bots (although that is fast changing with the advent of deep learning/machine learning based tools).

Some interesting (and unusual) CAPTCHA examples can be seen below:

- **HotCaptcha**: A briefly popular system which was shut down in 2009, verified 'humanness' by asking users to choose the top three most attractive individuals from a set of nine images.[3] (Figure 9)
- **HumanAuth CAPTCHA**: Also focused on image recognition but less widely adopted. Users were typically asked to solve tasks involving object recognition, such as identifying specific items in an image (Figure 10). It did not achieve the same level of widespread adoption as other image-based CAPTCHA systems like Google's reCAPTCHA. Its primary limitation was its inability to stay ahead of advancements in machine

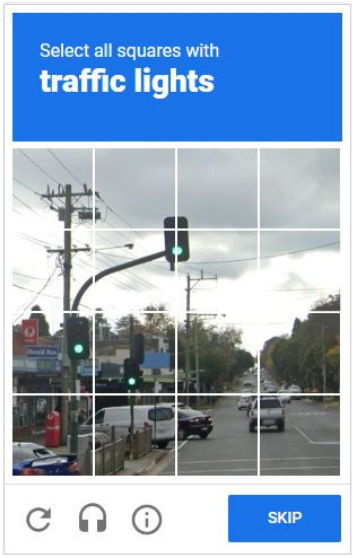

FIGURE 8. Example of an Image Based Captcha

FIGURE 9. Screenshot of what HotCaptcha looked like.

learning, where new AI models were increasingly capable of solving such image based challenges.[3]

VIII. Next Steps

In the upcoming weeks, we plan on addressing the following specific parts of our project:

- Completing the implementation of the attack for the Text Based CAPTCHA
- Implementing defenses (randomization) for our existing CAPTCHA systems
- Investigating case studies of CAPTCHA attacks (and their defenses)
- Investigating the implications of AI/ML on CAPTCHA security and design

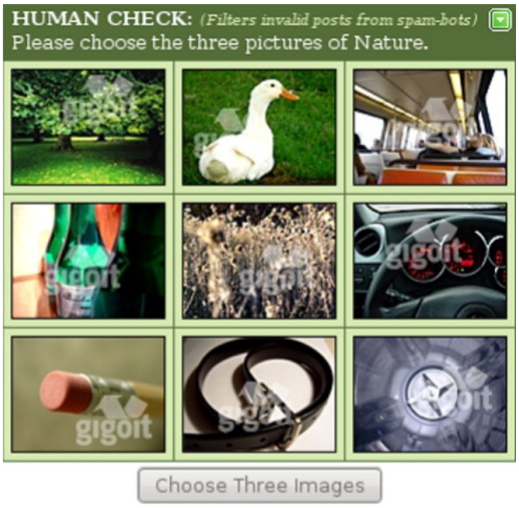

FIGURE 10. Screenshot of the HumanAuth CAPTCHA

- Evaluating factors that we considered/should be considered while designing a CAPTCHA to balance accessibility, security and computational resources.

References

[1] Stacey Burling. *CAPTCHA: The Story behind Those Squiggly Computer Letters*. Accessed: 2024-11-11. 2012. URL: https://phys.org/news/2012-06-captcha-story-squiggly-letters.html.

[2] Cloudflare. *What Is Credential Stuffing? — Credential Stuffing vs. Brute Force Attacks*. Accessed: 2024-11-11. 2024. URL: https://www.cloudflare.com/learning/bots/what-is-credential-stuffing/.

[3] Carlos Javier Hernandez-Castro and Arturo Ribagorda. "Pitfalls in CAPTCHA Design and Implementation: The Math CAPTCHA, a Case Study". In: *Computers & Security* 29.1 (Feb. 2010), pp. 141–157. DOI: 10.1016/j.cose.2009.06.006. URL: https://doi.org/10.1016/j.cose.2009.06.006.

[4] Aravinth Manivannan et al. "mCaptcha: Replacing Captchas with Rate Limiters to Improve Security and Accessibility". In: *Commun. ACM* 67.10 (Sept. 2024), pp. 70–80. DOI: 10.1145/3660628. URL: https://doi.org/10.1145/3660628.

[5] W3C. *Inaccessibility of CAPTCHA*. Accessed: 2024-11-11. 2024. URL: https://www.w3.org/TR/turingtest/.