The AI Hardness of CAPTCHAs does not imply Robust Network Security

Allan Caine and Urs Hengartner

University of Waterloo, Canada
{adcaine, uhengart}@cs.uwaterloo.ca

Abstract. A CAPTCHA is a special kind of AI hard test to prevent bots from logging into computer systems. We define an AI hard test to be a problem which is intractable for a computer to solve as a matter of general consensus of the AI community. On the Internet, CAPTCHAs are typically used to prevent bots from signing up for illegitimate e-mail accounts or to prevent ticket scalping on e-commerce web sites. We have found that a popular and distributed architecture for implementing CAPTCHAs used on the Internet has a flawed protocol. Consequently, the security that the CAPTCHA ought to provide does not work and is ineffective at keeping bots out. This paper discusses the flaw in the distributed architecture's protocol. We propose an improved protocol while keeping the current architecture intact. We implemented a bot, which is 100% effective at breaking CAPTCHAs that use this flawed protocol. Furthermore, our implementation of the improved protocol proves that it is not vulnerable to attack. We use two popular web sites, `tickets.com` and `youtube.com`, to demonstrate our point.

1 Introduction

A CAPTCHA is a special kind of AI hard test used to prohibit bots from gaining unauthorized access to web sites and computer systems. Using a definition similar to von Ahn *et al.* [1], we say that an AI problem is *hard* if it is the general consensus of the AI community that the problem is intractable when using a computer to solve it. CAPTCHAs are used by Yahoo! [2] to prevent bots from signing up for illegitimate e-mail accounts. Similarly, e-commerce web sites like the Minnesota Twins Major League Baseball Club [3] use CAPTCHAs to prevent ticket scalping by bots.

The word CAPTCHA stands for **C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part. Its basic operation is illustrated in Fig. 1. The central idea is simple: it is assumed that only humans can solve CAPTCHAs; bots cannot. There are two principals involved: the prover and the verifier. The verifier is an automated system. It generates a CAPTCHA image and evaluates the prover's response. If the prover's response is correct, the prover is admitted to the next step of the authentication process. If the prover's response is incorrect, the verifier bars the prover from proceeding any

Please use the following format when citing this chapter:

Caine, A. and Hengartner, U., 2007, in IFIP International Federation for Information Processing, Volume 238, Trust Management, eds. Etalle, S., Marsh, S., (Boston: Springer), pp. 367–382.

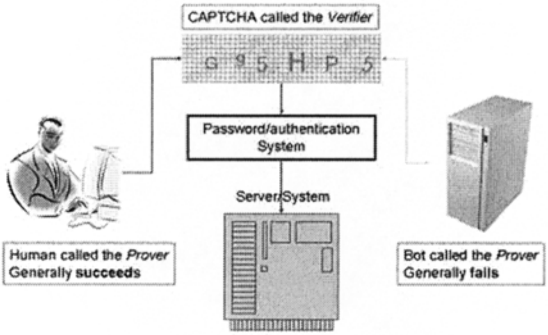

CAPTCHA called the *Verifier*
G 9 5 H P 5
Password/authentication System
Server/System
Human called the *Prover* Generally succeeds
Bot called the *Prover* Generally fails

Fig. 1. The verifier issues a visual test to the prover. In general, only human provers can solve CAPTCHAs.

further. If the prover is a human, the prover will generally succeed in solving the CAPTCHA; if the prover is a bot, the bot will generally fail.

There exists a popular architecture used by web sites that use CAPTCHAs for security. In this architecture, the security task is distributed amongst two servers: the Sales Server and the CAPTCHA Server. The Sales Server is responsible for the conduct of the e-commerce sales transaction; the CAPTCHA Server for generating the CAPTCHA image. This distributed approach is used so that many Sales Servers can utilize a single CAPTCHA Server.

In this paper,

- we show that the current protocol used in this architecture is insecure;
- we propose an improved and secure protocol while preserving the current distributed architecture;
- using a bot that we implemented, we prove that the current protocol is indeed insecure and subject to attack; and
- we prove that our implementation of our proposed protocol is indeed effective against the same attack.

The authors von Ahn *et al.* [1] suggest that a good CAPTCHA must be AI hard. Our research shows that their suggestion must be qualified. True, an AI hard CAPTCHA is a necessary condition but it is not a sufficient condition for robust network security. If the protocol is set up improperly, the CAPTCHA can be broken by an attacker with greater ease all things being equal. The problem rests with what we call a repeating CAPTCHA. Repeating CAPTCHAs are discussed in Sect. 2.

Our paper is organized as follows: Sect. 2 discusses the popular architecture and its insecure protocol. We show that the insecurity is the result of a repeating CAPTCHA. Section 3 discusses the attack, which exploits the insecurity identified in Sect. 2.

In Sect. 4 we propose a new and secure protocol. Our proposed protocol eliminates the repeating CAPTCHA. However, the current architecture is preserved.

Our experimental results are given in Sect. 5. It consists of three major subsections: the experimental results from our bot's attack on e-commerce web sites using a major U.S. ticket selling agent as our example; a demonstration of our implementation of our proposed protocol; and a discussion of `youtube.com`'s insecure protocol.

Section 6 discusses related work, and Sect. 7 sets out our conclusions.

2 Current Protocol

The current protocol is given in Fig. 2. It is used by web sites that employ CAPTCHAs for security and it involves three entities: the Sales Server, the CAPTCHA Server, and the Client. We learned of this protocol by examining HTML source code using `tickets.com` and `youtube.com` as our primary examples.

$$\text{Sales Server} : \text{Chooses random solution } s \tag{2.1}$$

$$\text{Sales Server} \rightarrow \text{Client} : E_c(s||\text{ID}||\,\text{MAC}_h(s||\text{ID})) \tag{2.2}$$

$$\text{Client} \rightarrow \text{CAPTCHA Server} : E_c(s||\text{ID}||\,\text{MAC}_h(s||\text{ID})) \tag{2.3}$$

$$\text{CAPTCHA Server} : \text{Generates CAPTCHA image with solution } s \tag{2.4}$$

$$\text{CAPTCHA Server} \rightarrow \text{Client} : \text{CAPTCHA image} \tag{2.5}$$

$$\text{Client} \rightarrow \text{Sales Server} : s', E_c(s||\text{ID}||\,\text{MAC}_h(s||\text{ID})) \tag{2.6}$$

$$\text{Sales Server} : \text{Proceed if } s = s' \wedge \exists \text{ ID} \tag{2.7}$$

Fig. 2. The current and popular protocol

The Sales Server is responsible for processing the sale, selecting a solution for the CAPTCHA image, and evaluating the Client's response. The CAPTCHA Server is responsible for generating the CAPTCHA image. The Client is the purchaser. The servers share a secret called c, which is used in a symmetric encryption function $E_c(\cdot)$ such as AES in CBC mode with a random initialization vector; and a shared secret h, which is used in a message authentication code $\text{MAC}_h(\cdot)$ such as HMAC [4]. There is a pre-existing session identifier ID. The servers trust each other, because the introduction of any distrust between the servers would undermine their effectiveness in providing the intended security. Finally, we note that the session ID is encrypted; otherwise, an attacker could build a database that would map IDs to CAPTCHAs and their solutions with the view to an on-line attack on the Sales Server.

If $s = s'$, the Sales Server allows the sale to proceed; otherwise, the sale is prohibited. The sale is also prohibited if the message from the Client to the Sales Server has expired. The message expires when the session ID expires. Fig. 3 shows the protocol graphically. The numbers correspond to the transaction numbers in Fig. 2.

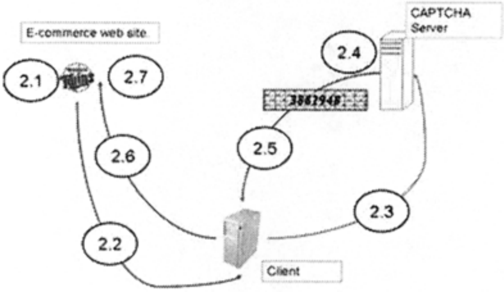
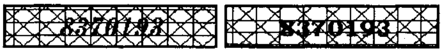


Fig. 3. Diagram of the current protocol

There is a flaw in message (2.3). An attacker can repeatedly send the message to the CAPTCHA Server, because the CAPTCHA Server does not keep state. The CAPTCHA Server is unaware that it has previously seen message (2.3). Each time the CAPTCHA Server receives message (2.3) from the Client, the CAPTCHA Server responds with a new CAPTCHA image.

Repeatedly sending message (2.3) generates a set of similar CAPTCHAs. We say that two CAPTCHAs are similar if they have the same solution, but they differ in terms of the transformation used. Fig. 4 illustrates two similar CAPTCHAs. The CAPTCHAs in Figs. 4(a) and 4(b) both have the solution 8370193, but each is rendered in a different font and a different background. We define a CAPTCHA Server which can be made to produce a set of similar CAPTCHAs a repeating CAPTCHA. We show in Sect. 3 that a repeating CAPTCHA places the attacker in a very advantageous position.



(a) First instantiation of 8370193 (b) Second instantiation of 8370193

Fig. 4. Two similar CAPTCHAs.

3 Attack

There are two steps in the attack: 1) collecting a representative sample of the characters used in the CAPTCHA and; 2) downloading a set of similar CAPTCHAs by repeatedly sending message (2.3) to the CAPTCHA Server and looking for patterns across that set of images.

We take tickets.com, a major U.S. ticket agent, as our example. They use CAPTCHAs to prevent ticket scalping by bots. The characters that are used in their CAPTCHAs are the digits zero to nine. Before we start running our attack, we download a number of CAPTCHAs and cut out the digits until a representative for each digit is found. Such a set is depicted in Fig. 5. These representative digits are called templates. Fig. 5 shows the templates after the noise has been removed by visual inspection on a commercially available photo editor. The templates are said to be clean. The templates are stored for re-use.

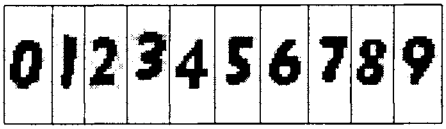
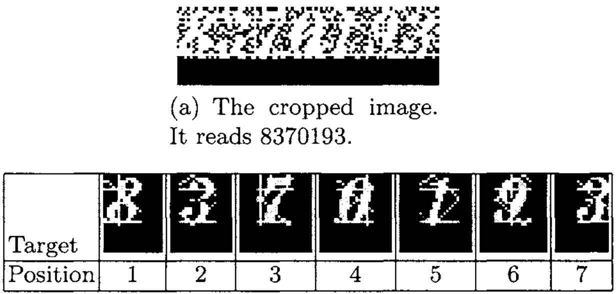


Fig. 5. Clean Templates

Once clean templates have been generated, the attack itself can begin. The bot downloads from the CAPTCHA Server a CAPTCHA image such as the one depicted in Fig. 4(a). Using a heuristic, the bot crops back the image as shown in Fig. 6(a). Next, the digits need to be segmented from each other.



(a) The cropped image. It reads 8370193.

(b) The seven target images produced from Fig. 6(a)

Fig. 6. The character segmentation process.

Since the digits are proportionally spaced, it is not possible to segment the digits by simply dividing up the image shown in Fig. 6(a) into equal segments along its width. Rather, the segmentation is done using k-means clustering [5] with the centroids equally spaced across the width of the image in Fig. 6(a). This segmentation produces seven target images as shown in Fig. 6(b).

The last step is to use the normalized cross correlation [5] to recognize the digit itself. We apply the normalized cross correlation, which gives us a score, S, each time we compare a template to a target image. The score is computed as

$$S = \max_{(u,v)} \left\{ \frac{\sum_{x,y} \left(I(u-x, v-y) - \bar{I}_{u,v}\right) \hat{T}(x,y)}{\sum_{x,y} \left(I(u-x, v-y) - \bar{I}_{u,v}\right)^2 \sum_{x,y} \hat{T}(x,y)^2} \right\} \tag{3.1}$$

where $\hat{T}$ is the template, I is the target, and $\bar{I}_{(u,v)}$ is the local average. The local average means the average of all of the pixels of I falling under $\hat{T}$ taking (u, v) as the upper left-hand corner of the template.

For example, if we compare the ten templates against a target image that actually contains a six, we get the scores shown in the bar chart of Fig. 7. As can be seen, template six obtains the best score. So, the target image would be judged to be a six.

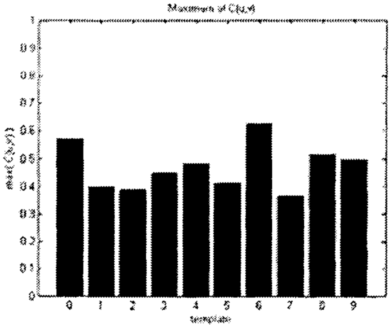


Fig. 7. The correlation scores for matching each of the templates 0 to 9 against a target known to contain a 6.

Yet, this method is not perfect. Sometimes a target image may be misinterpreted. For example, 3's are similar to 8's; 1's are similar to 7's; and 5's are similar to 6's. Also as can be seen in Fig. 6(b), the target images contain noise, which may adversely affect the correlation results.

Even so, the attack is not thwarted. By sending $E_c(s||\text{ID}||\,\text{MAC}_h(s||\text{ID}))$ to the CAPTCHA Server again, a similar CAPTCHA image can be downloaded as illustrated in Fig. 4(b). Through every iteration, tallies are kept of the best interpretations. A sample final result is given in Fig. 8. Voting for more than one possibility in any given character position is evidence of occasional misinterpretation. For example, in the Position 1 histogram given in Fig. 8, we can see voting for the 6 and the 7, although most of the votes were given to the 6 — the correct interpretation. Since there is a clear favorite interpretation in each of the seven positions, an attacker can determine the correct solution to the CAPTCHA. In Fig. 8, the correct solution is 6674846.

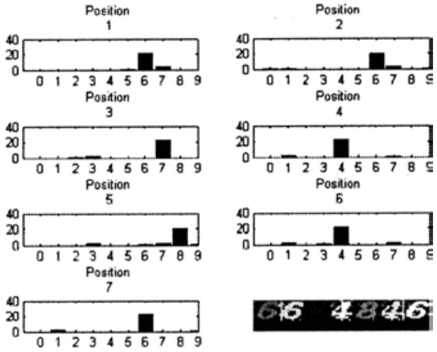


Fig. 8. The voting results on CAPTCHA "6674846"

4 Proposed Protocol

Essentially, the current protocol has one major downfall: the CAPTCHA Server depends upon the Sales Server to determine the solution to the CAPTCHA image. The attacker exploits this by sending message (2.3) repeatedly to the CAPTCHA Server. The attacker collects a set of similar CAPTCHA images, which she uses to break the CAPTCHA. The problem is cured by reassigning responsibilities. The CAPTCHA Server determines the solution instead of the Sales Server. Our proposed protocol is given in Fig. 9.

$$\begin{array}{rll}
\text{Sales Server} \rightarrow \text{Client} : & E_c(\text{ID}||\,\text{MAC}_h(\text{ID})) & (4.1)\\
\text{Client} \rightarrow \text{CAPTCHA Server} : & E_c(\text{ID}||\,\text{MAC}_h(\text{ID})) & (4.2)\\
\text{CAPTCHA Server} : & \text{Chooses solution } s\text{, and generates a CAPTCHA} & \\
 & \text{image with that solution} & (4.3)\\
\text{CAPTCHA Server} \rightarrow \text{Client} : & \text{CAPTCHA image}, E_c(s||\text{ID}||\,\text{MAC}_h(s||\text{ID})) & (4.4)\\
\text{Client} \rightarrow \text{Sales Server} : & s', E_c(s||\text{ID}||\,\text{MAC}_h(s||\text{ID})) & (4.5)\\
\text{Sales Server} : & \text{Proceed if } s = s' \wedge \exists \text{ ID} & (4.6)
\end{array}$$

Fig. 9. Our Proposed Protocol.

We make largely the same assumptions as we do in Sect. 2: there is a symmetric encryption function $E_c(\cdot)$ using a shared secret c; a message authentication code $\mathrm{MAC}_h(\cdot)$ using a shared secret h. The variable s is the chosen solution; s' is the Client's attempt at the solution. There is a pre-existing session identifier.

To determine if the client has passed or failed the CAPTCHA test, the Sales Server confirms the message's authenticity and integrity. If $s = s'$ and the Session ID returned by the CAPTCHA Server is the same as the current Session ID, then the Client passes the CAPTCHA test; otherwise, the Client fails. For the sake of clarity, we show the protocol in diagrammed form with the numbers in Fig. 10 corresponding to the message numbers in Fig. 9.

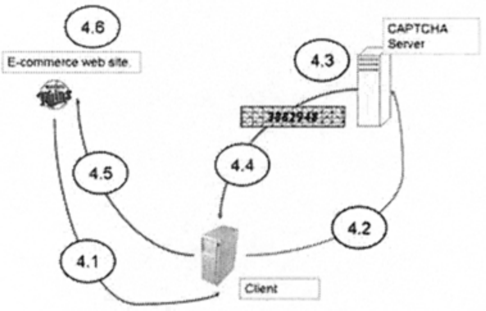


Fig. 10. Diagram of the proposed protocol

As pointed out earlier in Sect. 2, it is imperative that the ID be encrypted. Otherwise, the attacker can off-line query the CAPTCHA server for CAPTCHAs, solve them, build a database that maps IDs to CAPTCHAs and their solutions, and use this database in an on-line attack on the Sales Server

5 Experimental Results

This section consists of three major subsections. The first subsection discusses our attack. We prove that the security vulnerability in Sect. 2 truly exists. The second subsection demonstrates our implementation of our proposed protocol mentioned in Sect. 4 to show that the attack can be defeated. The third subsection discusses `youtube.com`'s repeating CAPTCHA and the security vulnerability it implies.

5.1 Attacking tickets.com

In our experiments designed to attack `tickets.com`, we wanted to find the answers to the following questions:

1. Is the attack as mentioned in Sect. 3 a realistic way of attacking CAPTCHAs?
2. Can the attack be conducted with a high probability of success?
3. If the attack is largely successful, does it place all of the clients of `tickets.com` in jeopardy or just some of them?

We built a bot to test our ability to break `tickets.com`'s CAPTCHA. We found that

1. It took on average 9.89 seconds and 7.2 queries to the CAPTCHA Server to break the CAPTCHA.
2. The attack was 100% successful.
3. All of the clients of `tickets.com` are at risk.

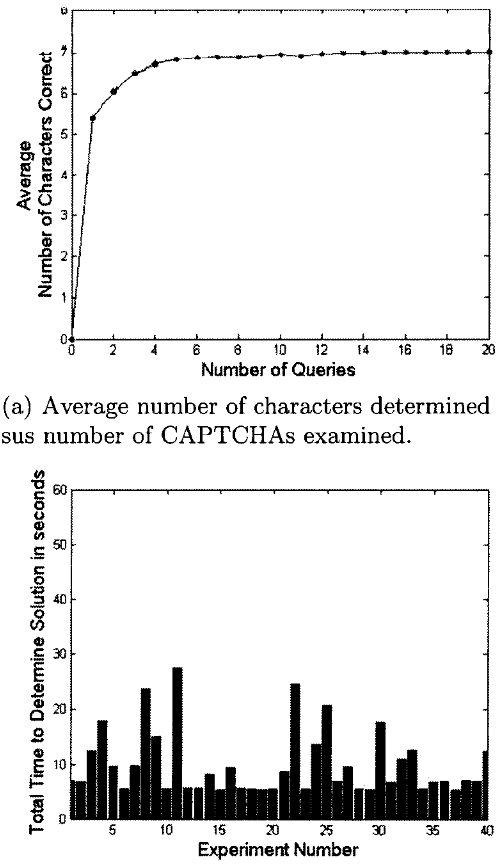
Setup of the Experiment For ethical reasons, we did not actually attack `tickets.com`'s clients directly over the Internet. Rather, we downloaded 20 different CAPTCHAs with identical solutions for each of the 40 experiments we conducted. As it turned out, downloading 20 images for each experiment was generally more than necessary. On average, only the first 7.2 images were needed by the bot to break the CAPTCHA.

The images were stored on our computer. Each image was given an index number reflecting the image's download order. Our bot strictly obeyed this ordering when fetching the images for processing.

Our bot ran on a Pentium 4 running at 3.2 GHz. The bot was written in the Matlab programming language. We used MATLAB Version 7.0.4.365 (R14) Service Pack 2 together with the statistics and image processing toolboxes. We used the statistics toolbox to have access to the k-means function, and the image processing toolbox for access to the image read function. The data we used can be found on our web page [6]. This web page periodically refreshes to reveal a new set of similar CAPTCHAs.

Our Bot's Success Rate and Running Time We ran 40 simulated attacks. They were all successful taking an average of 7.2 queries to the CAPTCHA Server. The minimum number of queries was 4; the maximum 20. Our results are summarized in Fig 11(a). It shows the average number of correct characters in all 40 attacks versus the number of queries made to the CAPTCHA Server. After one query, the bot knows 5.4 characters on average. After ten queries, the bot knows 6.925 characters on average with 38 out of the 40 experiments solved correctly. After examining 15 CAPTCHAs, our bot has determined the solution in all cases but the 11th. In retrospect, our bot had actually determined the correct answer in experiment 11 after examining 15 CAPTCHA images but it decided to increase its confidence in its answer by examining the remaining five CAPTCHAs.

While we were impressed with the bot's 100% success rate, we wanted to ensure that the bot was breaking the CAPTCHA in a reasonable period of time. It is the case that `tickets.com` allows the client only 60 seconds to solve the CAPTCHA. Our bot must break the CAPTCHA within that time limit.



(a) Average number of characters determined versus number of CAPTCHAs examined.

(b) Time to download and process images

Fig. 11. Experimental Results $N = 40$

The bot's average processing time is 7.85 seconds implying that 52.2 seconds are left to download on average 7.2 images. Each image is about 2.3 kB. Based on downloading 800 images, our experimental results show that it takes on average 0.2836 seconds to download one image. So, it would take 2.0419 seconds on average to download 7.2 images, which is far less time than the 52.2 seconds available.

Finally, we took the actual time reported by the bot to process the images and added 0.2836 seconds for each image that the bot reported having had processed. Our results are illustrated in Fig. 11(b). The average time to both download and process the images is 9.89 seconds, well within the 60-second time limit. Even in the worst case, the total time taken in experiment 11, including an estimate of network time, is 27.51 seconds. We claim that if the average download time of 0.2836 seconds per image prevails in an actual direct attack, our bot would succeed in breaking every CAPTCHA.

Risk to ticket.com's Clients Our experiments show that when a Client is making a purchase through `tickets.com`, the Client always calls a script located at `http://pvoimages.tickets.com/buy/NVImageGen` to fetch the CAPTCHA image. Our data set [6] is quite broad. It covers Major League baseball games, rock concerts, circuses, and children's entertainment to name a few. While it is true that the name of the e-commerce web site is passed to the CAPTCHA Server through the URL, this information is not used in determining the CAPTCHA image. As illustrated in Fig. 4(a), `tickets.com`'s CAPTCHAs are always characterized by a seven-digit number written in some font with a mesh of lines behind it. It is our view that our attack would succeed against any e-commerce web site supported by `tickets.com`.

5.2 Implementation of our Proposed Protocol

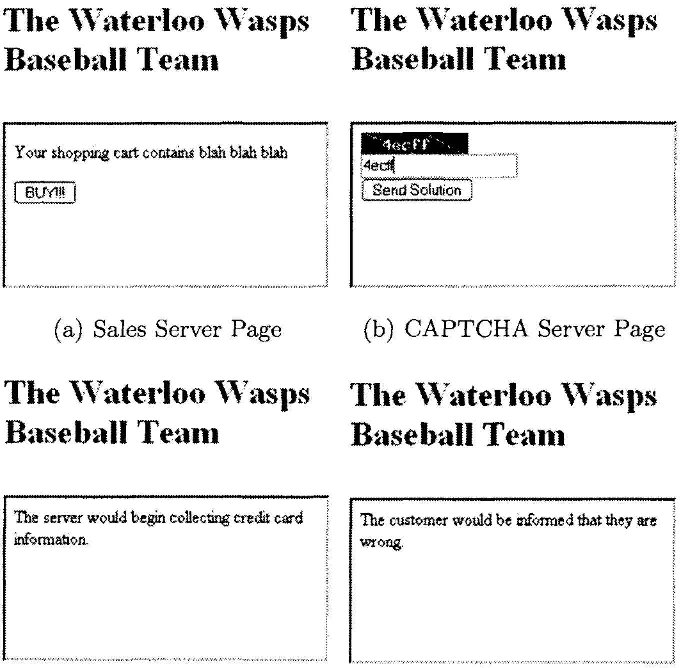
To demonstrate that our proposed protocol works, we implemented it as if it were being used on an e-commerce web site with anti-scalping security. We assumed that the Client had already selected her purchases and was ready to place her order.

We wrote two scripts in php: `SalesServer.php` and `CAPTCHAserver.php`. Each script emulates the roles of the Sales Server and CAPTCHA Server respectively. To avoid confusing the client as she moves from server to server, we used an embedded frame (iframe). In HTML, an iframe is essentially a browser within the Client's main browser window. The servers' output is directed to the iframe — not to the main browser window itself. Consequently, as the servers take turns responding to the Client's input, the change in servers is not reflected in the Client's address bar. From the Client's perspective, she would see herself as always being on the Sales Server's web page albeit with dynamic content. On the other hand, we admit that if the Client's browser does not support iframes, then the Client would be able to see the change in her browser's address bar.

Fig. 12(a) shows the opening page on `wrapper.html` [7]. At this point, message (4.1) of Fig. 9 is sent. The text inside the beveled border is actually code produced by `SalesServer.php` within the iframe. In practice, the beveled border would not normally be visible to the Client. The beveled border is being shown for the sake of clarity.

When the Client clicks on the BUY!!! button shown in Fig. 12(a), messages (4.2), (4.3), and (4.4) of Fig. 9 are sent. In Fig. 12(b), the HTML form shown within the beveled border is produced by the `CAPTCHAserver.php` script. Yet, the Client's address bar indicates that she is still on `wrapper.html`. So, while we have preserved the distributed architecture, we made it invisible to the Client.

The client enters her response to the CAPTCHA image and clicks on the SEND SOLUTION button shown in Fig. 12(b). With this mouse click, message (4.5) of Fig. 9 is sent. As illustrated in Fig. 12(c), if the Client enters the



The Waterloo Wasps Baseball Team

Your shopping cart contains blah blah blah

(a) Sales Server Page

The Waterloo Wasps Baseball Team

(b) CAPTCHA Server Page

The Waterloo Wasps Baseball Team

The server would begin collecting credit card information.

(c) Success

The Waterloo Wasps Baseball Team

The customer would be informed that they are wrong.

(d) Failure

Fig. 12. The HTML Pages from our implementation of our proposed protocol

correct solution, she receives an affirmative message and credit card information would now be taken from the Client. If the Client enters the wrong solution, the Client receives a negative indication from the Sales Server as in Fig. 12(d). Of course, in an actual implementation, the Sales Server would do more than simply post pass or fail messages in the window of the Client's browser.

If the Client should attempt to run the CAPTCHA Server script directly, the `CAPTCHAserver.php` script will detect that either message (4.2) is phony or non-existent. In either case, the script redirects the Client's browser to `wrapper.html`. Since the default of `wrapper.html`'s iframe is the Sales Server, the redirect is tantamount to compelling the Client to always go to the Sales Server first before going the CAPTCHA Server. The Client must follow the protocol. Unlike the current protocol, steps in our protocol cannot be circumvented or skipped to the attacker's advantage. They must be done in order from first to last.

Alternatively, even if an attacker should succeed in circumventing the message authentication, the script unconditionally chooses a new and random solution. The algorithm is given in the right-hand column of Fig. 13. The CAPTCHA will never repeat.

As earlier indicated in Sect. 5.1, at least four similar CAPTCHAs are needed by our bot to defeat `tickets.com`'s CAPTCHA. Our attack would not succeed against a non-repeating CAPTCHA. So, our attack has been defeated and `tickets.com`'s security vulnerability fixed using our proposed protocol. Yet, it remains to be seen if their CAPTCHA could be defeated without depending upon a repeating CAPTCHA; `tickets.com`'s CAPTCHA may still be vulnerable to other kinds of attacks.

Figure 13 gives the two php scripts as pseudo code. In the interests of clarity, we have left out the message authentication steps. We use c for the shared secret, and ID for the session identifier. The `SalesServer.php` script keeps state. Keeping state can be justified because the Sales Server needs keep track of the merchandise in the Client's electronic shopping basket anyway. On the other hand, `CAPTCHAserver.php` is stateless.

```
SalesServer.php

- Open the session
- if $\nexists$ ID
   - Generate session identifier ID
   - Echo out an HTML form with
   - $E_c$(ID) in a hidden field
   - a BUY!!! button
   - action attribute CAPTCHAserver.php.
- if $\exists$ ID
   - Compute $D_c(E_c(s))$
   - If $s = s'$ then admit the client; otherwise
     reject.
   - The script stops
```

```
CAPTCHAserver.php

- Choose a random $s$
- Compute $E_c(s)$
- Generate CAPTCHA image
- Echo out an HTML form with
   - $E_c(s)$ in a hidden field,
   - a text field for the client's solution ($s'$),
   - the CAPTCHA image,
   - a SUBMIT SOLUTION button
   - action attribute of SalesServer.php
```

Fig. 13. Algorithms for `SalesServer.php` and `CAPTCHAserver.php`. The HMAC steps have been omitted.

Hidden fields in the HTML forms are used for aesthetic reasons so that the form does not show the cypher text in the Client's browser window and possibly confuse the Client. It is not a security threat that the Client has a copy of the cypher text. If the Client attempts to alter the cypher text, the HMAC test of the server receiving the message will detect the alteration.

The advantage of our solution is that it maintains the existing architecture as closely as possible. As well, the distributed nature of the architecture is normally not apparent to the Client. On the other hand, we do admit that our proposed protocol requires two trips to the CAPTCHA Server: one trip to fetch the iframe and a second trip to fetch the CAPTCHA image. In the current protocol, only one trip is necessary. In addition, the CAPTCHA image must be of the form *uniqueName*.jpeg; some background process must generate those unique file names. Also, a background process must periodically clear away any old image files.

5.3 youtube.com

A new and popular web site for personal video sharing is called `youtube.com`. Our research shows that they too have a repeating CAPTCHA. They leave themselves vulnerable to attack. Their vulnerability seems to stem from a bug in their CAPTCHA server. To get their CAPTCHA to repeat, it is a simple matter of clicking on the "Can't read?" link soon after the sign up page loads [8]. Clicking the "Can't read?" link is analogous to sending message (2.3). Consequently, `youtube.com` has a repeating CAPTCHA.

Curiously, the window of opportunity eventually closes after a few minutes. Their CAPTCHA reverts from a repeating CAPTCHA to a non-repeating CAPTCHA. We suggest that `youtube.com` needs to examine their CAPTCHA server with a view to correcting this bug and resolving this security vulnerability.

6 Related Work

This paper focuses strictly upon text-based types of CAPTCHAs. However, there are other types of CAPTCHAs in existence. Examples of these other types can be found at The CAPTCHA Project [9].

We do not claim to be the first to have ever broken a CAPTCHA. It is unlikely that we will be the last. An extensive list of broken CAPTCHAs can be found at PWNtcha [10].

A major criticism of visual CAPTCHAs is that they are difficult if not impossible for the visually impaired to use. This point is brought up by Fukuda *et al.* [11]. From the authors' report, it does not appear that there currently exists any adequate solution to this problem without compromising security.

Mori and Malik [12] provide a detailed discussion regarding how they broke two other CAPTCHAs: GIMPY and EZ-GIMPY. [9] Our approach differs from theirs in that while they are looking at shape cues, we looking at correlation-based matching. They used tests to hypothesize the locations of characters while we used k-means clustering. Since GIMPY and EZ-GIMPY use English words, Mori and Malik could use that fact essentially as a conditional probability to determine the likelihood of the existence of a particular letter given the neighboring letters. On the other hand, we had no such similar advantage. The appearance of a digit in one location did not suggest the likelihood of a particular digit appearing in another location.

We also found it interesting that Mori and Malik [12] had a copy of the EZ-GIMPY and GIMPY software. Consequently, they could generate an unlimited number of CAPTCHA images. It is our contention that this kind of unlimited access can be a CAPTCHA's undoing. Indeed, our attack succeeded in part because we had virtually unlimited access to the CAPTCHA server at `tickets.com`. Yet, for us, we broke `tickets.com`'s CAPTCHA in spite of not being able to see their code.

Another ingenious way to solve CAPTCHAs is through free porn [13]. The user is enticed into the site, but the user's progress is occasionally blocked. The site presents the user with a CAPTCHA to be solved. However, the user is actually solving a CAPTCHA on an unrelated site. The attacker can then break the CAPTCHA on the other unrelated site.

There is quite a range of opinion on what constitutes success in breaking a CAPTCHA. The authors von Ahn *et al.* [14] suggest a success rate nearly as good as a human, while the W3C suggest a success rate as little as 10% [11]. Mori and Malik [12] declared success over GIMPY, the more difficult version of EZ-GIMPY, with a success rate of only 33%. We suggest that these differences in opinion stem from each author's implied threat model. For example, in our particular case, we suggest that a scalper needs a success rate near 100%, because the scalper must be able to buy up tickets quickly as soon as they go on sale. Otherwise, the scalper may be stuck with a small handful to tickets, which have not affected the market price and which are worth little more than their face value.

Finally, we agree fully with von Ahn *et al.* [14] that researching and breaking CAPTCHAs is a win-win scenario for both the AI community and for practitioners in network security. For the AI community, this research is profitable in the study of computer vision and object recognition. For the network security community, this research is beneficial in terms of designing better access control measures, which use AI as a means of telling humans and computers apart.

7 Conclusions

In this paper, we have shown that it is a security flaw to make the CAPTCHA Server dependent upon an outside entity to determine the solution for a CAPTCHA. This kind of protocol may lead to a repeating CAPTCHA. A repeating CAPTCHA may place the attacker in an advantageous position. We have also shown that it is important that web sites which employ CAPTCHAs ensure that no bugs exist in their scripts, which might cause the CAPTCHA to repeat even for a period of time.

We both proposed and implemented a protocol which can resist the outlined attack. We discovered that the attack is one which can succeed against any customer of `tickets.com`. This happens because all of `tickets.com`'s customers use the same CAPTCHA server.

We argue that our results are important in terms of the issues of trust and assurance. For example, in the context of ticket selling, a seller will not use a web site if the seller believes that she will expose herself to ticket scalping. Buyers, on the other hand, will become disillusioned with a web site if all of the best tickets are generally unavailable for sale. Companies like `tickets.com` must protect its principals from ticket scalping through the use of authentication protocols like CAPTCHAs.

Yet, for a CAPTCHA to be useful, it must be AI hard. In this paper, we have shown that while AI hardness is a necessary condition, it is not a sufficient condition for having a good CAPTCHA. A poorly implemented CAPTCHA can be AI softened; it becomes relatively easy to break. We have shown that a CAPTCHA that can be made to repeat itself is insecure. The attacker can use the numerous examples as a kind of sanity check before offering a response.

Acknowledgements

The authors wish to thank the reviewers for their helpful comments and feedback. We extend our thanks to Ian Goldberg for his suggestion to use embedded frames in our implementation, and to Richard Mann for his comments and suggestions regarding the computer vision portion of our paper.

References

1. von Ahn, L., Blum, M., Langford, J.: Telling humans and computers apart automatically. Commnications of the ACM **47**(2) (2004) 57 – 60
2. Yahoo! Inc.: Yahoo e-mail sign up. `http://www.yahoo.com` (2007)
3. Minnesota Twins Major League Baseball: Minnesota twins electronic ticketing. `http://minnesota.twins.mlb.com/` (2007)
4. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-hashing for message authentication. Internet RFC 2104 (1997)
5. Lewis, J.P.: Fast template matching. Vision Interface (1995) 120 – 123
6. Caine, A., Hengartner, U.: Data set. `http://www.cs.uwaterloo.ca/~adcaine/php/demo.htm` (2007)
7. Caine, A., Hengartner, U.: Implementation of proposed protocol. `http://www.cs.uwaterloo.ca/~adcaine/php/wrapper.html` (2007)
8. Youtube: Sign up page for youtube.com. `http://www.youtube.com/signup` (2007)
9. The CAPTCHA Project at Carnegie Mellon University. `http://www.captcha.net/` (2006)
10. PWNtcha captcha decoder. `http://sam.zoy.org/pwntcha/` (2006)
11. Fukuda, K., Garrigue, M.A., Gilman, A.: Inaccessibility of CAPTCHA. W3C (2005)
12. Mori, G., Malik, J.: Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA. In: CVPR. Volume 1. (2003) 134–141
13. Doctorow, C.: Solving and creating captchas with free porn. `http://boingboing.net/2004/01/27/solving_and_creating.html` (2004)
14. von Ahn, L., Blum, M., Hopper, N., Langford, J.: CAPTCHA: Using hard AI problems for security. Eurocrypt (2003)