CAPTCHA Smuggling: Hijacking Web Browsing Sessions to Create CAPTCHA Farms

Manuel Egele
Technical University Vienna, Austria
+43 58801-18318
manuel@iseclab.org

Leyla Bilge, Engin Kirda
Institute Eurecom, Sophia Antipolis, France
+33 4 93 00 81 00
{bilge,kirda}@eurecom.fr

Christopher Kruegel
University of California, Santa Barbara
+1 (805) 893-6198
chris@cs.ucsb.edu

ABSTRACT

CAPTCHAs protect online resources and services from automated access. From an attacker's point of view, they are typically perceived as an annoyance that prevents the mass creation of accounts or the automated posting of messages. Hence, miscreants strive to effectively bypass these protection mechanisms, using techniques such as optical character recognition or machine learning. However, as CAPTCHA systems evolve, they become more resilient against automated analysis approaches.

In this paper, we introduce and evaluate an attack that we denote as *CAPTCHA smuggling*. To perform CAPTCHA smuggling, the attacker slips CAPTCHA challenges into the web browsing sessions of unsuspecting victims, misusing their ability to solve these challenges. A key point of our attack is that the CAPTCHAs are surreptitiously injected into interactions with benign web applications (such as web mail or social networking sites). As a result, they are perceived as a normal part of the application and raise no suspicion. Our evaluation, based on realistic user experiments, shows that CAPTCHA smuggling attacks are feasible in practice.

Categories and Subject Descriptors

H.M [**Information Systems**]: Miscellaneous; D.2.0 [**Software**]: Software Engineering : General

General Terms

Security threats

Keywords

CAPTCHA, attack, real-world experiments

1. INTRODUCTION

Completely Automated Public Turing tests to tell Computers and Humans Apart (CAPTCHAs) [18] are often the first line of defense in many online services. Their purpose is to protect such services and resources from automated misuse by malicious programs. For example, free email providers frequently use CAPTCHAs to prevent spammers from automatically creating disposable email addresses that can then be used for spam campaigns. Similarly, blogging and forum sites make use of CAPTCHAs to prevent bot-generated postings. These postings aim to lure readers into following links that may point to malicious pages where drive-by attacks or other dangerous content might await them.

Note that CAPTCHAs may also be used to limit automated access to search engines. For example, the Google search engine requires a visitor to solve a CAPTCHA once it detects abnormal search behavior [13] (e.g., a high number of search queries stemming from a specific IP).

Reliably breaking CAPTCHAs enables attackers to create large numbers of fake email accounts, or simplify their spamming activities on blog and forum sites. As the mail servers of the major web-based email providers (e.g., Microsoft LiveMail, Gmail, or Yahoo) are commonly not included in spam blacklists, such accounts are valuable to spammers. In fact, there was a large increase of spam messages originating from LiveMail accounts immediately after its CAPTCHA system was reported to be broken [22].

Ever since CAPTCHAs have been used to prevent automated access to specific online services, miscreants have tried to circumvent these protection mechanisms. The possible attack strategies are manifold, and include techniques that rely on optical character recognition (OCR), mechanisms to identify distorted characters [24], machine learning techniques [6] to break CAPTCHAs, and human "farms" where real people manually solve the challenges [14].

In this paper, we present a novel attack that we denote *CAPTCHA smuggling*. In a CAPTCHA smuggling attack, user interactions with legitimate online services (such as web mail or social networking sites) are intercepted by the attacker (i.e., a malicious program executing on the victim's computer) and put on hold until the victim solves a CAPTCHA challenge. The displayed CAPTCHA and its surrounding browser window spoof the visual characteristics of the online service that the victim is using. Hence, it is difficult for victims to distinguish between real CAPTCHAs displayed by the online service and CAPTCHAs smuggled into the session by the attacker. As the CAPTCHA challenge is under the direct control of the attacker, a malicious program that needs to solve a CAPTCHA can forward the challenge to a victim's computer. The malicious component on this computer then performs the CAPTCHA smuggling attack (and thus, gets the challenge solved by an unsuspecting user). The premise of our attack is that users are so accustomed to solving CAPTCHAs while using online services that they will not notice extra CAPTCHAs that are smuggled in by a malicious application running on their computer.

SAC'10 March 22-26, 2010, Sierre, Switzerland.

Note that malicious activity related to solving CAPTCHAs has already been seen on the the Internet. Troj/CAPTCHA-A [17], for example, displays a series of pictures of a woman who takes off her clothes. In order to see the next picture, the user has to solve a CAPTCHA. Similarly, certain adult web pages require a user to solve a CAPTCHA before the actual content is displayed [7]. However, the inventors of CAPTCHAs [18] do not consider this "pornography attack" against CAPTCHAs a concern as annoyed visitors can easily switch to other offers. In contrast to such an attack, *CAPTCHA smuggling* is performed seemingly as part of (popular) online services that the victim has been using over a period of time. Thus, it is unlikely that, given a low enough level of annoyance (a low number of CAPTCHA puzzles per day), a user would choose a different online service.

The typical attack scenario that we envision involves a botnet with bots that intercept user interactions and smuggle CAPTCHAs into the victim's active web browsing sessions. For example, a Facebook CAPTCHA that is under the attacker's control would sometimes be displayed when the victim starts to compose a message or send a friend request. Requiring a victim to solve only a few CAPTCHAs a day ensures that the manipulation stays unnoticed and is perceived as normal procedure. Note that a CAPTCHA smuggling attack is very lightweight in terms of required resources. Therefore, it is trivial for the bot master to add the required functionality to the existing bot program without limiting the existing functionality of the botnet.

To test the feasibility of our attack, we conducted real-world user experiments. The results of these experiments suggest that CAPTCHA smuggling is feasible in practice and can be used by attackers to make victims solve CAPTCHA challenges on their behalf.

This paper makes the following contributions:

- We introduce the *CAPTCHA smuggling* attack, and we describe the implementation of a man-in-the-middle component that performs such attacks.
- We report on a bug in Firefox and a problem with Facebook that made it easier for us to distribute our attack prototype in a stealthy fashion.
- We describe the results of our real-world experiments indicating that CAPTCHA smuggling attacks are feasible in practice. Based on these results we give an estimate how many CAPTCHAs a botmaster could solve using this attack.

The remainder of this paper is structured as follows: Section 2 gives an overview of existing CAPTCHA systems. Section 3 introduces the technique of *CAPTCHA smuggling*. Section 4 describes our prototype implementation. The setup for our user experiments is detailed in Section 5, while the results of the experiments is presented in Section 6. Mitigation approaches against CAPTCHA smuggling attacks are described in Section 7. Section 8 discusses research that is related to our work. Finally, Section 9 concludes the paper.

2. A BRIEF OVERVIEW OF CAPTCHAS

A CAPTCHA is a challenge-response test used to determine whether the response is generated by a computer or a human. These tests are designed to be easily solvable by humans, but difficult to decipher for automated programs.

Text-based CAPTCHAs are the most common. These consist of a sequence of distorted characters rendered into an image. Bending, rotating, or mutating colors further complicates the task for OCR programs [12] to accurately identify the characters. A popular state-of-the-art, text-based CAPTCHA is reCAPTCHA [21]. The tests used by reCAPTCHA are derived from the attempt to digitize old books. Clearly, words that cannot be recognized during scanning have already circumvented sophisticated OCR techniques. reCAPTCHA makes use of these words and forms a challenge by combining an unknown word with a control word whose content is known. To further thwart programs that try to solve the challenge, the words are distorted and aligned randomly. reCAPTCHA accepts a solution if the control word is submitted correctly, and the text for the unknown word overlaps substantially with already submitted solutions for the same challenge.

Recent advances in CAPTCHA systems resulted in *image-based* CAPTCHAs. Asirra [4], is an image-based CAPTCHA that requires the user to distinguish between images of cats and dogs. An Asirra challenge consists of 12 images, each showing either a cat or a dog. A solution is accepted as correct if the user successfully selects all the cat pictures, but none of the dog images. The authors argue that the underlying computer vision problem [5] is particularly difficult to solve efficiently.

Current CAPTCHA systems such as reCAPTCHA suggest that automatically breaking CAPTCHAs will become much more difficult in the near future. Nevertheless, attackers are constantly trying to automatically break CAPTCHAs using botnets, and have succeeded in breaking them in many cases (e.g., [22]). As botnets are already used to break CAPTCHAs, we believe that the next step for attackers are CAPTCHA smuggling attacks where CAPTCHAs are injected into legitimate web browsing sessions of victims.

3. CAPTCHA SMUGGLING ATTACKS

To perform a successful CAPTCHA smuggling attack, a malicious component (such as a bot program) on the victims' host needs to intercept the user interactions with an online service (e.g., Facebook) and delay their execution until the victim successfully solved a CAPTCHA challenge. Bear in mind that it is not necessary to create a new botnet for CAPTCHA smuggling attacks. Existing bot programs can easily be extended to perform such attacks in addition to their current behavior. This process is depicted in Figure 1.

In a typical attack, the user first performs an action that the attacker wishes to delay (for example, sending a request to a specific web server, or clicking a button on a web page). The malware on the victims' host intercepts the request and locally stores all information necessary to replay the request later on. This component then retrieves a CAPTCHA challenge from the attacker's server. The attacker would forward a challenge that needs to be solved in order to perform the desired action. He could, for example, forward a CAPTCHA that was encountered during the registration of an email account. Once the user has solved the challenge, the malware replays the intercepted request from the stored information. To the unsuspecting victim, it seems as if the web application she is using is protected by a standard CAPTCHA mechanism. In reality, however, the victim has just provided the attacker with the necessary information to continue his nefarious tasks.

In the remainder of this section, we discuss the design of our prototype system that performs *CAPTCHA smuggling* attacks. In order to have a flexible solution, we implemented a plugin for the popular Mozilla Firefox web browser. Table 1 lists the web sites that our prototype targets and the user interactions that it intercepts (e.g., composing a new message).

Note that all web sites (online services) listed in Table 1 require that a user solves at least one CAPTCHA during account registration. Moreover, the social networking site Facebook displays CAPTCHAs on different occasions even after an account has been

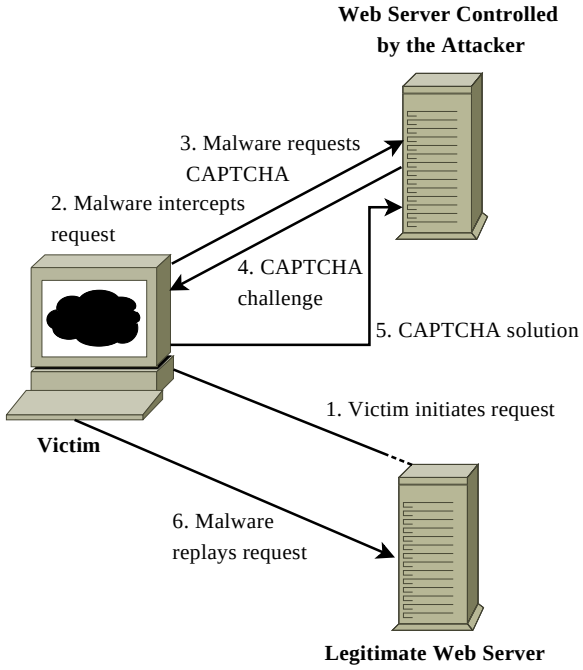

Figure 1: CAPTCHA Smuggling

Facebook: Login, open Facebook application, send message, change profile settings, comment on status messages, post to wall
Microsoft LiveMail: Send email
Twitter: Follow tweet
Flickr: Comment picture
Gmail: Send email

Table 1: User interactions that trigger CAPTCHAs

successfully created. For example, to become friends with another user on Facebook, one has to send a so-called friend request, asking that user for permission to add her to one's friend list. Sending many consecutive friend requests typically provokes a CAPTCHA challenge, as Facebook tries to prevent automated friend requests and spamming. Only after that challenge is solved, the request is permitted. In our attack, we smuggle CAPTCHA challenges into an active web browsing session by creating an additional `iframe` node in the DOM tree of the document, thus tricking the victim into solving our challenge.

4. PROTOTYPE IMPLEMENTATION

Firefox supports extensions as a means for third party developers to extend the existing browser functionality. Popular examples include plugins that block advertisements on web pages, or extensions that block the execution of all script content embedded in web sites, unless they are specifically permitted by the user.

Extensions for the Mozilla line of products are commonly written in JavaScript. For Firefox, this implies that the rich API that the browser exports to the scripting engine is readily available to the plugin developer. Modifying page contents, for example, is just as easy as intercepting mouse clicks to elements (e.g., buttons, links, etc.) on a web site. The extension that we implemented makes use of this functionality.

The actions that should trigger CAPTCHA challenges can be defined in two ways in our prototoype: (1) By uniquely identifying an HTML element whose click event is hooked, or (2) by specifying a regular expression that matches the URL of HTTP requests:

The identification of an HTML element can happen by specifying the values of attributes, such as `id` or `class` properties. Once a page is finished loading and rendered, the DOM tree of the current document is searched for elements that match the given specification. For each element that matches the description, the plugin registers an additional click handler that intercepts the click event. To intercept HTTP requests that match the specified regular expressions, the plugin compares all outgoing requests with this pattern. If the request matches the specification, the request is put on hold and the CAPTCHA smuggling code is invoked instead.

When a certain user action triggers a CAPTCHA challenge (as discussed in the previous paragraph), this action is intercepted. That is, the plugin discards the event and keeps a record of the intercepted action. This information is necessary to replay the action once the CAPTCHA challenge is solved. Then, the plugin adds an HTML `iframe` tag to the DOM tree of the current document, displaying a CAPTCHA challenge that is retrieved from our server. To avoid raising any suspicion, the CAPTCHA challenge mimics the target web site challenge as closely as possible. To this end, we downloaded the HTML and CSS sources that define the dialog windows of the targeted web sites (services) and modified them accordingly. In addition, the label of the submit button is modified to resemble the intercepted action. If the CAPTCHA challenge is solved successfully, the plugin replays the previously intercepted action from the stored information, which is then executed without further interference.

Once installed, the plugin applies several techniques to hide its presence on the infected system. First, the plugin removes itself from the list of installed add-ons in Firefox. That is, the common method of uninstalling the plugin via the user interface is not possible anymore. Second, for the first hour after its installation, the plugin does not show any CAPTCHAs. We chose to implement this feature as we believe that victims might get suspicious if, right after installing a new plugin, they would see more CAPTCHA requests then usual. Third, the plugin contains an adjustable threshold that controls the frequency at which CAPTCHAs are displayed to the victim. Initially, we set this value to display CAPTCHAs for only 15% of the monitored actions.

5. DISTRIBUTING THE PLUGIN TO REAL USERS

As we did not have a botnet at hand to test-drive our implementation, and as such an experiment would have been unethical, we had to distribute the plugin via other channels to test the feasibility of our idea. To this end, we recruited volunteers among our friends who would deliberately install the plugin. Besides assuring the volunteers that no sensitive information (e.g., login credentials) are stolen, we did not give them any further information regarding the functionality of the plugin. Furthermore, we could convince some of these volunteers to post links on Facebook that point to a page that performs a social engineering attack. More precisely, the page would ask its visitors to install our plugin, with the premise that this plugin is needed to see a video.

In addition to links and pictures, Facebook allows to share video clips with friends. During the sharing process, the user selects a thumbnail frame that represents the content of the video. This frame is then overlayed with a play button and shared among the users' friends. To watch the video, other users just have to click

the overlayed frame. If the proper video codec is missing (e.g., no Flash plugin present), the video is replaced by a message informing the user of this situation. In addition, this message contains a button that allows to install the required plugin. In our attacks, we used this standard Facebook behavior to disguise our social engineering attack. That is, we created a web page that imitates the above mentioned message to lure a visitor into installing our plugin. In the following, we elaborate on two problems that we discovered in Facebook and Firefox, respectively, that allowed us to increase the credibility of this page.

Cloaking Facebook: Cloaking is a technique commonly applied in the field of search engine optimization [23]. This term refers to a method where search engine spiders are served with different content than real visitors, thus making the page appear in search results for queries for which it does not provide any information. Cloaking helped us to disguise the link to our plugin as a video. This was achieved as follows:

Whenever a link to a resource is posted on a Facebook profile, the Facebook server fetches a copy of that resource. This copy, which is stored on the Facebook server, is then displayed as a thumbnail of that resource. The HTTP user-agent header field that is used in this request[1] identifies Facebook as the origin. Based on the user-agent identifier, a real visitor can be distinguished from the Facebook server, and different content can be presented to each one. Therefore, we were able to return a fake image, mimicking a video resource, to the Facebook server. When a real user requests the resource, however, we redirect her to the page that attempts to install the plugin. Once the user installs the plugin, her browser is instructed to load the real movie.

Firefox disguises the origin of the plugin: Once a user is convinced that she wants to see the video and clicks the fake video thumbnail, the following happens: Instead of opening the content right away, Facebook loads a page with additional controls that allow the user to comment on, or share the content. The content-page itself is embedded via an `iframe` tag. In our case, this content-page tries to install a browser plugin. This behavior triggers a security warning in Firefox and requires the user to agree to install that plugin. However, Firefox does not identify the source of the plugin correctly. In fact, the notification indicates that the source of the plugin is *www.facebook.com*, while in fact the true source was our own server (see Figure 2). We notified the Firefox developers of this behavior, who confirmed that this is unintended. Of course, we added an appropriate bug report to the Mozilla bug tracking system.

Clearly, one question that arises is if it is ethically acceptable and justifiable to conduct such experiments with real users (especially considering the social engineering aspects). Similar to the experiments conducted by Jakobsson et al. in [8, 9], we believe that realistic experiments are the only way to reliably estimate success rates of real attacks. Hence, we chose to test our attacks on real users to evaluate the potential of CAPTCHA smuggling attacks. Our work is in accordance with Jakobsson et al.'s definition of ethical fraud experiments. That is, we do not expose the participants of our experiment to any risk. Furthermore, our CAPTCHA smuggling system does not attempt to steal any sensitive information from the user and consumes minimal additional resources in terms of bandwidth, CPU cycles, and time of the user. Thus, the negative impact on any infected victim is very low.

Of course, we included functionality that allows us to notify the user of the plugin's existence after the experiment. Since the plugin requests the CAPTCHA challenge from our server, we can simply substitute the challenge with a message that informs the user. Furthermore, the plugin has the capability to reverse its removal from the list of installed add-ons. This allows the user to uninstall the plugin from the Firefox add-on dialog.

[1] facebookexternalhit/1.0
(+http://www.facebook.com/externalhit_uatext.php)

6. EVALUATION

In this section, we present the evaluation of the data we gathered during our experiments. We began to distribute the plugin to volunteers on May 15th 2009. Within 14 days, we counted 17 successful installations. In addition, five volunteers agreed to post a link to our fake video page in their Facebook profile. This resulted in another 7 people who installed the plugin. In total, our fake video page was visited 56 times and we could achieve 24 successful installations of our plugin. Note that two of these 24 users seem to use Firefox only on a non-regular basis. That is, they first tried to access the plugin with an unsupported web browser (i.e., not Firefox). Only after the page indicated that the content is only available to Firefox users, these users revisited the site using the Firefox browser. Since, despite a successfully installed plugin, neither of these two installations ever requested a CAPTCHA, the assumption that these users rely on an other browser for every day work seems justified.

Of these 24 installations, 17 requested CAPTCHA challenges from our server. The remaining 7 never requested any challenges. That is, these users did not perform any of the monitored interactions often enough to exceed the frequency limits and be presented with a CAPTCHA. A possible explanation is that these users only rarely use the online services we are monitoring with our plugin. In addition, users that do not produce content, such as commenting or posting to the service, will not see any of our CAPTCHAs. Furthermore, if a user makes use of the auto-login feature of the web site, our system will not display CAPTCHAs during the login procedure.

In total, these 17 infected users requested 167 CAPTCHAs. 126 were solved successfully. For 9 challenges the users submitted a wrong solution, and in 32 instances, the users did not submit any solutions (e.g., canceling the action). Note that a user failing to submit any solution to a challenge does not pose a problem for the attacker. The reason is that the same CAPTCHA can be forwarded to another victim, once a timeout expires during which no result is received. On average, a CAPTCHA was solved within 11 seconds. Table 2 lists the monitored actions that triggered requests for CAPTCHA challenges. Although we implemented the CAPTCHA smuggling for many different actions and online services, only the Facebook and Gmail actions resulted in requests to our CAPTCHA server during our experiments.

Action	# solved	# failed
Facebook login	68	2
Facebook Post to Wall	6	1
Facebook Open an Application	6	2
Facebook Send Message	5	0
Facebook Comment Wall-Post	35	4
Facebook Change Profile Settings	1	0
Gmail Send Email	5	0
Total	126	9

Table 2: Actions that triggered CAPTCHA requests

Table 2 indicates that most of the solutions submitted in response to the spoofed CAPTCHA challenges were correct. In our experiment, we could observe an overall success rate, in terms of correctly solved CAPTCHAs, of 75%. Assuming that unanswered

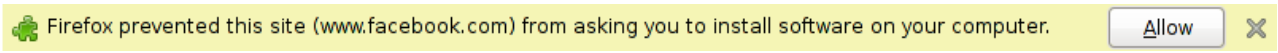

Figure 2: Firefox warning indicates wrong origin of plugin

CAPTCHA challenges are forwarded to other victims this value increases to 93%.

During our experiments, we gradually increased the frequency at which the user is prompted to solve CAPTCHAs. The rationale behind this was to learn at what level the users would get annoyed and just cancel the action instead of submitting solutions to the CAPTCHA challenges. From an attacker's point of view, this value denotes the maximum output she can expect from performing a CAPTCHA smuggling attack. We varied the frequency between 15%, 25%, 35%, and 50%. Interestingly, not even setting the frequency to 50% resulted in a noticeable reaction by the users. That is, even if the users are required to solve a CAPTCHA for every second monitored action, they still submit valid solutions. Table 3 presents a detailed breakdown of the gathered data. As one would expect, the number of solved CAPTCHAs per day and user is directly proportional to the frequency at which the plugin displays CAPTCHA challenges.

Freq.	days	# users	# correct solutions	# CAPTCHAs per day per user
15	6	10	30	0.5
25	3	5	24	1.6
35	3	10	57	1.9
50	2	3	15	2.5

Table 3: CAPTCHA smuggling success rates

Discussion.

Our premise is that the attacker already controls a botnet, and hence, has access to the victim's computer to perform CAPTCHA smuggling that happens in addition to the already existing bot behavior. Our experiment shows that even without such a powerful infrastructure, an attacker can achieve a considerable number of infections. In particular, exploiting the trust relationships between users on social networking sites together with social engineering could allow an attacker to successfully infect many users. For example, extending our system with the ability to automatically propagate the fake video content on an infected profile would have been straightforward. Clearly, we did not wish to create a plugin implementing worm-like propagation strategies in Facebook.

Also, we note that our experiment mainly focused on Facebook. Of course, a real attacker could extend CAPTCHA smuggling to other online services as well.

To roughly estimate the monetary gain an attacker can expect from performing CAPTCHA smuggling attacks, we refer to the Symantec Internet Threat Report for the year 2008 [16]. According to these studies, email credentials are the third most frequently offered information in the underground economy. The lowest estimated value for such credentials is listed at 0.10$. Taking a medium sized botnet of 10,000 infected machines as a basis and assuming that each user can be tricked into solving only two CAPTCHAs per day, this would result in 2,000$ daily revenue only for email account credentials. A botnet of this size is easily possible, as recent studies [15] indicate that current botnets can reach sizes of hundreds of thousands of infected machines.

7. MITIGATION APPROACHES

Although an attacker controlling a victim's machine via a bot program is a powerful enemy, this section introduces techniques that could improve current CAPTCHA systems to make them more resilient against CAPTCHA smuggling attacks. As a first improvement, we suggest CAPTCHA systems that make use of additional forms of authentication towards the user. *Site Keys*, commonly applied in online banking systems to thwart phishing attacks, can easily be adapted to raise the bar for attackers to perform CAPTCHA smuggling attacks. This technique uses visual information as a shared secret to indicate to the user that the content she is viewing is indeed from the site she is currently interacting with. To this end, the user can select a custom image that is included on the content produced by the service in question (e.g., an online banking login site authenticates itself to the user by including a user-selected image). In terms of CAPTCHA systems, this scheme can be applied by letting the user choose the background for all CAPTCHAs. The user is then instructed to solve only such CAPTCHAs that have the chosen background set. A CAPTCHA challenge retrieved by the attacker and forwarded to an infected user would not have the correct background set, and the user can easily detect the attempted attack.

Note that it is not impossible for an attacker to circumvent this additional protection. In theory, a sophisticated attacker could spoof the Site Key as well. However, this simple addition considerably raises the bar to successfully launch CAPTCHA smuggling attacks.

Recall that the main purpose of CAPTCHAs is to tell computers and humans apart. Hence, if an online service can tell with certainty that the current user is human, such tests are not required anymore. Identifying users with information that is not as easily available as email addresses fulfills this requirement. Facebook, for example, allows its users to register their phones with the service. Users who perform this registration can post news to their profiles from their phones and, additionally, are exempt from solving CAPTCHAs. Educating users that registering their phone numbers will prevent further CAPTCHA challenges would make it easy for the victims to recognize potential CAPTCHA smuggling attacks.

8. RELATED WORK

This section covers related work in the field of CAPTCHAs and automated attempts on breaking them. Von Ahn et al. were the first to introduce CAPTCHAs in the year 2000 [19]. Their later work [20] elaborates on different techniques that can be used to tell computers and humans apart automatically. Chew et al. focused on challenges that are based on image recognition [2]. They propose a system where the a human user should describe the subject in a picture, or recognize an interfering image from an otherwise coherent set of pictures. Making CAPTCHAs usable on mobile devices is the main contribution of [3] by Chow et al. Their system does not rely on keyboard input, which can be annoying especially on mobile devices. Instead, they designed a CAPTCHA that can be solved with touch screens or numeric keypads.

Kolupaev et al. [10] summarizes automatic techniques that break text-based CAPTCHAs by applying OCR methods. Mori et al. [11] perform shape context matching allowing them to break the EZ-Gimpy CAPTCHA. This variant of a text-based CAPTCHA consists of approximately seven words that are cluttered in an image.

The challenge is solved if three words in such a cluttered image are correctly recognized.

The Asirra CAPTCHA, introduced by Elson et al. [4] relies on a mechanism where the user has to identify pictures of cats out of a set of twelve pictures of cats and dogs. The authors argue that this distinction is hard to make for computers even under different attack scenarios. This assumption is partially falsified by Golle [6]. By leveraging machine learning techniques, it was possible to break this scheme with a probability of 10.7%.

Among the heavy users of CAPTCHAs are social networking sites. These sites prevent automated crawling attacks by requiring the user to solve CAPTCHAs every once in a while. By targeting social networks for impersonation attacks, Bilge et al. [1] demonstrated that the employed CAPTCHA techniques are not sufficient to prevent such automated attacks. The main reason for their success in automatically breaking CAPTCHAs is that the system allows to request an arbitrary amount of CAPTCHAs for the same challenge. Introducing a reasonable threshold for the maximum number of CAPTCHAs that can be requested would make such automatic attacks much more difficult.

Although many of the automated attacks have more or less reasonable success rates (up to 10%), it is very unlikely that the rates will remain at this level once current CAPTCHA systems are upgraded and improved. As CAPTCHAs are intended to be solved by humans (only), *CAPTCHA smuggling* allows to have constantly high success rates even for advanced CAPTCHA systems that successfully thwart automated attacks.

9. CONCLUSION

Improvements of automated attacks against current CAPTCHA techniques drives the development of more robust CAPTCHA techniques. Once the computational effort of breaking CAPTCHAs reliably becomes too high, attackers could try to misuse unsuspecting victims to solve CAPTCHAs. *CAPTCHA smuggling* intercepts user interactions and presents the victim with a CAPTCHA the attacker needs solved to continue his malicious tasks. The victim is conned into believing that the CAPTCHA is displayed by the legitimate online service, and she has no easy way to distinguish between an authentic and a smuggled-in CAPTCHA.

We implemented a proof-of-concept plugin for the popular Firefox browser, and we evaluated our CAPTCHA smuggling attack with realistic real-world user experiments. Our results suggest that it is feasible for an attacker to launch such attacks in practice and achieve high volumes and success rates for solving CAPTCHAs.

Acknowledgments

This work has been supported by the Austrian Science Foundation (FWF) under grant P18764, SECoverer FIT-IT Trust in IT-Systems 2. Call, Austria, Secure Business Austria (SBA), and the WOMBAT and FORWARD projects funded by the European Commission in the 7th Framework.

10. REFERENCES

[1] L. Bilge, T. Strufe, D. Balzarotti, and E. Kirda. All your contacts are belong to us: automated identity theft attacks on social networks. In *WWW '09: Proceedings of the 18th international conference on World wide web*, pages 551–560, New York, NY, USA, 2009. ACM.

[2] M. Chew and J. D. Tygar. Image recognition captchas. In *Information Security, 7th International Conference, ISC*, pages 268–279, 2004.

[3] R. Chow, P. Golle, M. Jakobsson, L. Wang, and X. Wang. Making captchas clickable. In *HotMobile '08: Proceedings of the 9th workshop on Mobile computing systems and applications*, pages 91–94, New York, NY, USA, 2008. ACM.

[4] J. Elson, J. R. Douceur, J. Howell, and J. Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. In *ACM Conference on Computer and Communications Security*, pages 366–374, 2007.

[5] M. Everingham, A. Zisserman, C. K. I. Williams, and L. Van Gool. The PASCAL Visual Object Classes Challenge 2006 (VOC2006) Results. http://www.pascal-network.org/challenges/VOC/voc2006/results.pdf.

[6] P. Golle. Machine learning attacks against the asirra captcha. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 535–542, New York, NY, USA, 2008. ACM.

[7] Heise Online. Cracking Google captchas with porn. `http://www.heise.de/english/newsticker/news/113336`, 2008.

[8] M. Jakobsson, P. Finn, and N. Johnson. Why and how to perform fraud experiments. *Security & Privacy, IEEE*, 6(2):66–68, March-April 2008.

[9] M. Jakobsson and J. Ratkiewicz. Designing ethical phishing experiments: a study of (rot13) ronl query features. In *WWW '06: Proceedings of the 15th international conference on World Wide Web*, pages 513–522, New York, NY, USA, 2006. ACM.

[10] A. Kolupaev and J. Ogijenko. Captchas: Humans vs. bots. *IEEE Security and Privacy*, 6(1):68–70, 2008.

[11] G. Mori and J. Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2003), 16-22 June 2003, Madison, WI, USA*, pages 134–144, 2003.

[12] S. Mori, C. Y. Suen, and K. Yamamoto. Historical review of OCR research and development. *Document image analysis*, pages 244–273, 1995.

[13] N. Provos. Google online security blog: The reason behind the "we're sorry..." message. `http://googleonlinesecurity.blogspot.com/2007/07/reason-behind-were-sorry-message.html`, 2007.

[14] B. Stone. Breaking google captchas for some extra cash. `http://bits.blogs.nytimes.com/2008/03/13/breaking-google-captchas-for-3-a-day/`, 2008.

[15] B. Stone-Gross, M. Cova, L. Cavallaro, B. Gilbert, M. Szydlowski, R. Kemmerer, C. Kruegel, and G. Vigna. Your botnet is my botnet: Analysis of a botnet takeover. Technical report, University of California, Santa Barbara, 2009.

[16] Symantec Corporation. Internet security threat report, volume XIV. `http://eval.symantec.com/mktginfo/enterprise/white_papers/b-whitepaper_internet_security_threat_report_xiv_04-2009.en-us.pdf`, 2009.

[17] Humans + porn = solved captcha. *Network Security*, 2007(11):2 – 2, 2007.

[18] C. M. University. The Official CAPTCHA Site. `http://captcha.net`.

[19] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha: Using hard ai problems for security. In *EUROCRYPT*, pages 294–311, 2003.

[20] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47(2):56–60, 2004.

[21] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *Science*, September 2008.

[22] Websense. Microsoft live hotmail under attack by streamlined anti-captcha and mass-mailing operations. `http://securitylabs.websense.com/content/Blogs/3063.aspx`, 2008.

[23] B. Wu and B. Davison. Cloaking and Redirection: A Preliminary Study. In *Adversarial Information Retrieval on the Web*, 2005.

[24] J. Yan and A. S. El Ahmad. A low-cost attack on a microsoft captcha. In *CCS '08: Proceedings of the 15th ACM conference on Computer and communications security*, pages 543–554, New York, NY, USA, 2008. ACM.