Tell Me About Yourself: The Malicious CAPTCHA Attack

Nethanel Gelernter
Dept. of Computer Science
College of Management Academic Studies
nethanel.gelernter@gmail.com

Amir Herzberg
Dept. of Computer Science
Bar Ilan University
amir.herzberg@gmail.com

ABSTRACT

We present the *malicious CAPTCHA attack*, allowing a rogue website to trick users into unknowingly disclosing their private information. The rogue site displays the private information to the user in obfuscated manner, as if it is a CAPTCHA challenge; the user is unaware that solving the CAPTCHA, results in disclosing private information. This circumvents the Same Origin Policy (SOP), whose goal is to prevent access by rogue sites to private information, by exploiting the fact that many websites allow *display* of private information (to the user), upon requests from any (even rogue) website. Information so disclosed includes name, phone number, email and physical addresses, search history, preferences, partial credit card numbers, and more.

The vulnerability is common and the attack works for many popular sites, including nine out of the ten most popular websites. We evaluated the attack using IRB-approved, ethical user experiments.

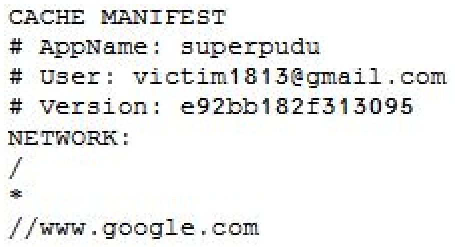
1. INTRODUCTION

Rogue websites exploit browser vulnerabilities, `con' the user (phishing, scams, malware,...), and perform *cross-site* attacks, to extract or manipulate user information at `victim' websites.

The main defense against cross-site attacks is the *Same Origin Policy (SOP)* access-control mechanism [22], implemented in all browsers. Grossly simpli ed, the SOP allows a script received from one site, say *rogue.org*, to access only objects and responses from sites in the same domain (*rogue.org*). The goal is to prevent rogue websites from learning information about the interaction of the user with other, *target* sites (which are from di erent `origin'), while allowing `legitimate' web use.

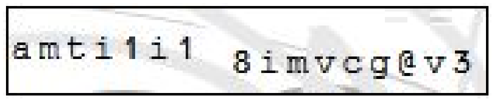
We present the *cross-site malicious CAPTCHA attack*, which circumvents the SOP to expose private details of the user, kept by the `victim' site. In contrast to XSS and other known attacks, the malicious CAPTCHA attack does not extract information from the (SOP-restricted) ow from site to browser, or depend on browser or website vulnerabilities.

WWW 2016, April 11–15, 2016, Montréal, Québec, Canada.
ACM 978-1-4503-4143-1/16/04.
DOI: shttp://dx.doi.org/10.1145/2872427.2883005.



```
CACHE MANIFEST
# AppName: superpudu
# User: victim1813@gmail.com
# Version: e92bb182f313095
NETWORK:
/
*
//www.google.com
```

(a) Gmail application cache manifest



(b) Anagram CAPTCHA of rst 15 characters

Figure 1: Exposing the rst 15 characters of the Gmail address of the victim (victim1813@gmail.com), using xed-width font anagram CAPTCHA

Instead, *the user* is tricked into providing her own information to the attacker, i.e., the attack exploits *the user as a side channel*.

Speci cally, the rogue site presents to the user what appears as an ordinary CAPTCHA. In reality, this is one or often multiple small frames containing private information of the user, embedded from the target website. The user, unaware of the fact that the CAPTCHA contains her own information, `answers' it, i.e., types and sends the contents to the attacker. This `user side-channel' can be similarly abused in other ways, e.g., displaying the private information as part of a game or typing-test. We focused on CAPTCHA since it is a generic, well-de ned and widely-used mechanism. Indeed, users are so used to being forced to solve CAPTCHAs, they usually just `solve' the CAPTCHA by lling in the required response, without paying much attention to the contents. See [15].

An example for a CAPTCHA that encapsulates the Gmail address of the victim appears in Figure 1.

The malicious CAPTCHA attack can only expose information that is both displayed by the target website to the user and allowed to be embedded in a di erent site (in a frame). Sites often restrict the presentation of their pages within frames (e.g., using the XFO header), mainly to defend against clickjacking attacks [24]; see Section 2.2. This limits the scope of information that can be included in the malicious CAPTCHA in a site-speci c way. However, Table 1 shows that all but one of the ten most-popular websites, and other important sites, allow the embedding of pages/objects

	Name	Additional information
Google & Youtube	√	Email address, followed users, liked items, Google+ circles
Facebook	√	Followed users, liked items
Yahoo	√	Anti-CSRF token
Baidu	√	Personal information: calendar, age, education, job, interests etc.
Wikipedia	√	
Amazon	√	Number of items in the cart, customer ID
Twitter		
Taobao	√	Physical address, phone number, birthday
QQ	√	Subject and details of recently received emails
Live & Bing & Outlook	√	Search history, Skype client ID
eBay	√	Physical address
Alibaba & Aliexpress	√	Email and physical addresses, birthday, phone number
Craigslist	√	Email address, saved searches
Booking.com	√	Credit card last digits and expiration

Table 1: Information vulnerable to the malicious-CAPTCHA attack for the ten most popular sites [2], and below, for four other important sites.

containing sensitive user information and are vulnerable to the malicious CAPTCHA attack. As shown, the malicious CAPTCHA may expose the name of the victim user, email and physical addresses, information about preferences, personal information provided by the user, search history, partial credit card details, and more. This information can be abused for phishing and other attacks. In particular, the attack exposes the Yahoo! anti-CSRF token, thereby facilitating CSRF attacks.

We validated the e ectiveness of our malicious CAPTCHA attacks in ethical, IRB-approved user experiments. In particular, we evaluated di erent obfuscation methods, such as shu ing the iframes and adding dummy data to the CAPTCHA, to make it even harder for users to detect the attack. In Section 4, we further develop the attack and show how to use a single malicious CAPTCHA to answer multiple questions about private data.

Ethics. We disclosed our attacks to the relevant websites to allow them to incorporate defenses. All of our usability experiments were IRB-approved.

Contributions

In this work, we make the following contributions:

- Introduce the malicious CAPTCHA attack and show that it exposes sensitive information about users from most popular sites. We also show that for some popular sites, it is possible to steal tokens that can be exploited for other attacks (see Table 1).
- Present di erent obfuscation methods for the malicious CAPTCHA attack.
- Suggest a variant of malicious CAPTCHA attacks for e ciently exposing multiple Boolean properties.
- Evaluate their e ectiveness using IRB-approved, ethical user experiments.
- Present defenses against our attacks.

Organization. In Section 2, we describe the adversary model and brie y provide the necessary background on iframes. In Section 3, we present and demonstrate several variants of the malicious CAPTCHA attack. In Section 4 we show how to use a single malicious CAPTCHA to answer multiple questions about private data, and demonstrate how the attacker can learn about the preferences of Facebook and Google users, and expose the search history of Bing users. In Section 5, we evaluate the attacks discussed in the previous sections. Before we conclude the paper, we dedicate two sections to defenses and related work.

2. PRELIMINARIES

2.1 Adversary Model and Roadmap

To launch the malicious CAPTCHA attack, only two modest capabilities are required from the attacker: (1) control one or more malicious web pages, and (2) to be able to `lure' victims into visiting these pages.

We present the attack procedure, assuming that JavaScript is enabled in the victim's browser. However, it is possible to create a malicious CAPTCHA without JavaScript. In this case, the attacker might present a CAPTCHA to a website for which the user is not authenticated.

Many attacks are possible under this adversary model. However, the most severe attacks, such as cross-site scripting (XSS) or installing malware, require nding vulnerabilities in websites or browsers. While it is considered very di cult to nd such vulnerabilities in popular browsers and websites like Gmail or Facebook, the malicious CAPTCHA can be applied on both of them.

Stealing information about the user can also be done via social engineering. However, unlike the malicious CAPTCHA attack, in social engineering the user is aware that she is giving away her details. Many users might deliver incorrect or ctitious details if they are asked to do so explicitly.

The roadmap of the malicious CAPTCHA attack contains three steps: (1) luring the victim to the attacker's malicious page, (2) determining the websites for which the victim is authenticated, and (3) manipulating personalized information of the victim from such a website and presenting it as a malicious CAPTCHA. We brie y describe the rst two steps; the third step is discussed in depth in the subsequent sections.

Luring the victim user to the attacker's webpage. The basic requirement of the malicious CAPTCHA attack, as well as any other cross-site attack, is to cause the user to visit the attacker's page. This requirement is considered easy and is the rst step of several other known attacks. To lure random users into the website, the attacker can use legitimate site-promotion techniques. For example, the attacker can advertise a free downloads website, and require the user to ll in a CAPTCHA before a le is downloaded. Other ways to lure many users, or even a speci c one, is using phishing emails and social-engineering techniques [12,18,19].

Determine target websites to which the user is currently logged-on. Because the malicious CAPTCHA attack relies on loading personalized pages of websites, it is crucial for the attacker to rst determine the websites to which the victim is logged-on. The adversary can use known cross-site login detection techniques [6,13,20,26]. Alternatively, for very popular web services such as Google and Facebook, the adversary can assume that the victim is logged-in and a posteriori verify the login status from the input of the victim to the malicious CAPTCHA.

2.2 Loading Webpages in iFrame and Clickjacking

All modern browsers allow a host webpage to include frames called *HTML iframes* from other `guest' websites. This is used in many legitimate ways. Since the Same Origin Policy prevents the host webpage from accessing the contents of the guest website, this feature is considered to be secure against leakage of information.

However, the use of frames is known to facilitate *clickjacking* [3, 14, 24] attacks. Clickjacking (also called UI redress attack, UI redressing), is a malicious technique that tricks its victim into unconsciously performing *actions* in a guest website. In clickjacking, the attacker uses CSS features to load a frame containing part of the guest webpage as an invisible, transparent layer, over a benign-looking link or button in the hosting webpage. The victim user is manipulated into clicking the benign-looking link or button shown, not knowing that she is actually clicking on a link or a button in the host website, possibly purchasing some product or unintentionally signaling `like' to some item (`Likejacking').

The most e ective and widely adopted countermeasure against clickjacking, is the *X-Frame-Options (XFO)* HTTP response header. This server-client defense was proposed and integrated into Internet Explorer by Microsoft in 2009 [10], and later implemented in the other popular browsers. Brie y, the XFO header is sent in HTTP response by the server, and instructs the browser to avoid loading the content of the response in an iframe.

3. THE MALICIOUS CAPTCHA ATTACK

The malicious CAPTCHA attack tricks web users who are used to solving CAPTCHAs into copying and sending information about themselves to the attacker, without realizing it is their personal information. Basically, the attacker loads personalized webpages with private information in several iframes, and uses CSS tricks to make the set of iframes appear as a CAPTCHA, without arousing the user's suspicion.

We assume that a victim user is visiting an attacking, rogue website, and that the victim is authenticated to a *target website*. This allows the rogue site to use an inline frame (iframe) to load some of the target-site pages containing private, personal details. We refer to the page containing the private information as the *target page*, and to the sensitive, private information that the attacker wants to expose, as the *target private record (TPR)*.

There are three challenges to the malicious-CAPTCHA attack: (1) The attacker needs to *identify an exploitable TPR* that can be loaded in a frame within the rogue webpage, in particular, not blocked by an appropriate XFO header. (2) The attacker needs to present the TPR in one or more frames, while *avoiding attack detection.*; the user may detect the attack due to the content (e.g., user recognizing her own name in the CAPTCHA), or due to di erences in style from typical CAPTCHA. (3) The attacker needs to *recover the TPR* from the user's response to the CAPTCHA, possibly dealing with user mistakes (e.g., using only lowercase) and possible presentation errors.

We rst explain how it is possible to identify exploitable TPRs in Section 3.1. In Section 3.2 we discuss *simple CAPTCHA*, which presents one frame containing the TPR. Simple CAPTCHA is limited to scenarios in which the user is unlikely to recognize the TPR. In Section 3.3, we discuss anagram obfuscation. In Section 3.4, we describe two advanced variants of the attack, and in Section 3.5 we present a detailed example. Finally, in Section 3.6, we discuss the technical aspects of transforming a TPR into a CAPTCHA to minimize the risk of raising suspicion while maximizing accuracy.

3.1 Identify Exploitable TPR

We followed a simple algorithm to identify private or sensitive information from a website that can be loaded in an iframe. We rst created an account for the website and created a list with private information or sensitive data that might be related to this account. This is actually a list of known TPR values that we would want to extract if our account was the victim. Next we crawled the website while the account was authenticated and recorded the tra c using an HTTP proxy. Finally, we searched for the list's TPRs in the HTTP responses that did not contain the XFO header. We also veri ed that the TPRs are indeed visible, to avoid cases where they appear in attributes or in JavaScript code that are not presented by the browser.

We also searched the TPRs in the browser using regular string searches. This served to overcome cases where a TPR is received in an HTTP response with XFO header, but is later presented in a page that can be loaded in an iframe (e.g., using AJAX).

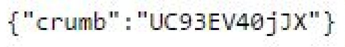
3.2 Simple CAPTCHA

The *simple CAPTCHA* uses a single iframe that contains and presents to the user the whole TPR as the CAPTCHA text, or some of the CAPTCHA text. This technique is practical only when the user is unlikely to recognize the fact that the TPR contains sensitive, private information. For example, using simple CAPTCHA to present a user-recognizable TPR, such as one containing the user's name, may cause users to suspect the rogue website of an attack. Surprisingly, we found that some TPRs contain sensitive, privacy-intrusive content, yet are unlikely to raise users' suspicions, even when presented directly to the user in a simple CAPTCHA. We o er two such examples.

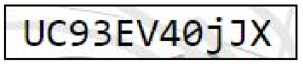
ID numbers. Many websites identify users, groups, and other items using unique *user ID numbers*. Most of the users do not know their user ID, or the ID of the items/pages they visit. However, detecting the ID of an item is usually equivalent to detecting the item itself, since mapping an ID to its item is usually a simple procedure.

Speci cally, we found that in Microsoft email service (outlook.com), it is possible to load a JSON le in an iframe, such that the Skype client ID of the user appears in a xed position relative to the beginning of the response. Similarly, it is possible to load the customer ID of Amazon users.

Anti CSRF tokens. Anti-CSRF tokens are sent together with HTTP requests to detect and prevent CSRF attacks [25]. Anti-CSRF tokens should be infeasible to forge or predict, and as such, they are usually generated pseudo-randomly. Most users are not even aware of the existence of these tokens. Nevertheless, an attacker who gains an anti-CSRF token of a user that already visits the attacker's website, can often launch a successful CSRF attack, and thereby perform operations as if it was authorized by the user.



{"crumb":"UC93EV40jJX"}



UC93EV40jJX

(a) Crumb received as JSON le (b) Simple anagram

Figure 2: Simple CAPTCHA based on Yahoo! anti-CSRF token (crumb)

Speci cally, Yahoo! uses several CSRF tokens, called *crumbs*. We found that it is possible to load a short JSON le that contains a crumb used in the Yahoo! Messenger application. This crumb is included with every message sent out by the Messenger application.

The original JSON le, with the crumb and the simple CAPTCHA that is based on it, appear in Figure 2.

3.3 Anagram CAPTCHA

An anagram is a word or phrase that is created by rearranging the letters of another word or phrase. For example, *Elvis* can be rearranged as *livEs*. In an *anagram CAPTCHA*, the attacker displays di erent parts of the TPR in tiny iframes, which we call *fragments*. The fragments are permuted from their original positions using a randomly chosen permutation, thereby creating an anagram. The attacker, upon receiving the permuted fragments from the victim, can reproduce the TPR.

The mere fact that the CAPTCHA contains an anagram of the text from the TPR, may su ce to make it less likely for the victim user to suspect the anagram CAPTCHA. Indeed, the anagram is essentially a transposition cipher, applied once using a randomly chosen key.

3.3.1 Fixed-width Font Anagram CAPTCHA

In a *fixed-width font anagram CAPTCHA*, the attacker displays each letter of the TPR in a tiny iframe, which we call a *fragment*.

Clearly, this type of CAPTCHA works best when the TPR is presented using a xed-width (also called monospace or non-proportional) font; namely, every character occupies the same amount of horizontal space. Most websites do not use monospace xed-width fonts. However, by default, Google Chrome, Mozilla Firefox, and the Safari browsers use monospace font to present HTTP responses that contain plain text (without HTML tags), e.g., JSON les. In particular, this holds for the examples presented above (in Section 3.2) of TPRs that can be used in a simple CAPTCHA.

There are other types of TPRs that can be presented using xed-width font CAPTCHA. For example, it is possible to load the Gmail address of a user in an iframe. The Gmail address appears as a remark in a xed position within an application cache manifest [23] that can be loaded in an iframe. Hence, the rogue website can use a xed-width anagram CAPTCHA to expose the user's Gmail address. See Figure 1 for an example of a manifest and the corresponding CAPTCHA.

3.3.2 Variable-width Font CAPTCHA

We found that most of the private data that can be loaded in an iframe, cannot be displayed using xed-width font. Using the method of Section 3.3.1 would not work well, since attackers cannot isolate each character into a separate iframe. In these cases, the attacker can create a *variable-width font anagram CAPTCHA*, which is basically a permutation of blindly cut pieces of the TPR.

The main challenge in creating such an anagram CAPTCHA is that with variable-width (proportional) fonts, each character takes up only as much width as required by the shape of the character; some are wider, some are narrower. For example, the letter l takes signi cantly less space than the letter m or even L.

When `blindly carving' a small iframe containing only a small part of the TPR, using variable-width font, the width of each letter depends on the letter itself. Because the letters are not known in advance, it is inevitable that some letters will be `cut'. Namely, it is impossible for the attacker to split the TPR into several fragments, each containing a whole - yet single - letter.

Assuming that each TPR letter should appear in some fragment, in its entirety, each fragment should be wider than the widest letter. Moreover, to ensure that even the widest letter is not cut, the width of the fragment should be twice as wide as the widest letter. However, for some fonts, using this value will result in fragments that are too wide; this may alert the user into noticing part of her private information, possibly exposing the attack. In our experimental validation (see Section 5.1), we used a fragment width that is a bit more than twice the width of most of the small letters.

Additionally, for every fragment width, simply cutting the TPR into disjoint fragments will probably result in cut letters. To facilitate reconstruction of the TPR, the fragments should have some overlap between them. Note that fragments will often contain partial, `cut' letters in their right-hand and/or left-hand edges. In our experiments, we instructed participants to ignore cut letters using a message placed below the CAPTCHA.

See Figure 4(a) for an example of variable-length font CAPTCHA, exposing the user's name (in this example, `Inno Cent'). The CAPTCHA contains six (permuted) iframes, each containing a part of the Facebook comments box (Figure 3).

3.4 Avoiding Detection and Improving TPR Recovery

The CAPTCHAs we described so far have two main weaknesses:

1. *Avoiding detection* may be a challenge, especially when the TPR is short, contains repeating characters, or is embedded using variable-width font (Section 3.3.2), where a single fragment might contain two or more consecutive characters.
2. *TPR recovery* may also be a challenge, especially when, due to variable-width font, fragments contain parts of characters and overlapping characters. Furthermore, the above methods do not provide any mechanism to detect (or correct) user-errors.

We now present two techniques that help overcome these weaknesses.

Camou aged CAPTCHA. The *camouflaged CAPTCHA* adds `dummy' characters to the CAPTCHA, to camou age the fact that the anagram uses the same characters as the victim user's TPR. This addition of irrelevant letters makes it even harder for the victim to detect the TPR. This is especially signi cant when the TPR is very short or contains very few characters.

Additionally, the attacker can use the `dummy' characters to verify that the victim's input re ects some of the challenge text presented, and to localize the characters. Unlike the

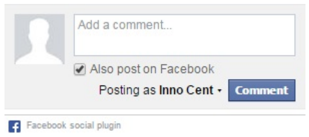
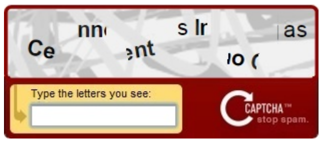
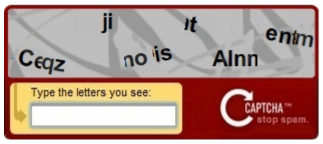


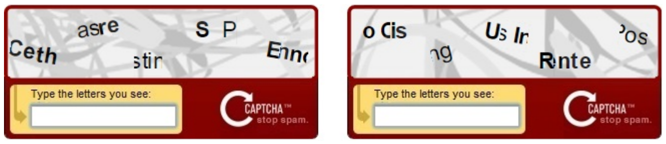
Figure 3: Facebook comments box for user Inno Cent



(a) anagram



(b) Camou aged anagram



(c) Two-step camou aged anagram

Figure 4: Three CAPTCHAs exposing the user name *Inno Cent* by embedding the Facebook comments box (Figure 3). The camou aged-anagram CAPTCHA is based on a comment box in which the name is adjusted left-to-right without any text after it. In the other CAPTCHAs, it is possible to see fragments of the phrase *Posting as*.

TPR, the attacker knows the dummy letters, and if they do not appear in the solution, it is an indication that the user might not have lled in the CAPTCHA correctly.

Two-step CAPTCHA. CAPTCHAs have become harder and harder to solve to prevent bots from cracking them. Web users have become accustomed to obfuscated CAPTCHAs that are di cult to solve, and to failures in solving them. The two-step CAPTCHA exploits this fact by separating the TPR into *two CAPTCHAs*, where each is an anagram (possibly camou aged) on di erent parts of the TPR. The attacker presents the rst CAPTCHA; once the victim completes it successfully, the attacker continues as though an incorrect solution was received, and replaces the rst CAPTCHA with the second. With two-step CAPTCHA, the whole TPR does not appear on the victim's screen at any one time. This makes the detection of the TPR harder. The two-step CAPTCHA also allows exposure of more parts of a long TPR and the use of camou age characters to detect errors and ease recovery.

3.5 Example: Exposing Facebook Username

Facebook prevents its pages from appearing in the iframes of other websites. However, its social plugins [11] allow other websites to integrate Facebook features. The comments box is one of these plugins. This box presents comments by Facebook users, allowing a logged-in Facebook user to comment. Because the name of the logged-in client appears in the comments box, an attacker can embed fragments of the comments box in malicious CAPTCHAs to extract the name of the user.

Personal names di er in their lengths; hence, in some cases a long name may not completely appear in the CAPTCHA. However, we found that the attacker can signi cantly control the characters displayed, using text width and language (text direction), e.g., to extract the rst or the last names as required. For example, if the attacker wants to get the last name, she should use a comments box in which the name is adjusted right-to-left (Figure 3). She can then create the fragments in this direction, beginning with the position of the last letter of the name. Figure 4 demonstrates the three techniques based on loading the Facebook comments box (Figure 3) in iframes.

3.6 Presenting Content as a CAPTCHA

We now explain how a rogue website can present the TPR so it looks like a typical CAPTCHA, and how to transpose the TPR as required for anagram CAPTCHAs. The challenge is the same origin policy (SOP), which does not allow direct access to the content of the target page, and in particular to the TPR. Instead, we use the fact that the rogue site can load the target page containing the TPR in an iframe, and manipulate its appearance. We now explain HTML/CSS techniques to create fake CAPTCHA from iframes of the target page, and discuss several challenges.

Cutting the TPR. We use CSS to present only a speci c part of the TPR, which we call a *fragment*. Speci cally, to load a fragment, we use a *div* element with the size of the fragment, and load the target page within the *div* element. Assume that in the target webpage, the position of the fragment is given as o sets of L pixels from the left of the page and T pixels from the top. Placing the iframe in o sets of $-L$ and $-T$ pixels to the left and the top of its default position in the div element, will present the fragment inside the div element. We use CSS to hide any part of the iframe that is presented outside the boundaries of its parent *div* element.

Controlling the size of the text. We use the scaling option of CSS to manipulate the size of the *div* element, so that its size is appropriate for use in CAPTCHA. In many websites, the TPR appears in tiny font, which is not usually used in CAPTCHA; hence, scaling is crucial.

CAPTCHA look and feel. Although we cannot control the style of the target page, it is possible to a ect its view by adding semi-transparent elements over it. This is important because the text's color and the background may help the user recognize the target page. In addition to the challenge text, CAPTCHAs often involve some lines on the text or in the background. We imitate this behavior by adding semi-transparent pictures of lines. It is also possible to rotate some of the fragments; this is another transformation visually similar to the transformations used in CAPTCHA.

Preventing access to the target webpage. Presenting a webpage in an iframe does not restrict the user from accessing it. Even if scrolling is disabled for the iframe, it is still possible to focus on the iframe and to move to other parts of the page. This may allow users to notice that this is not a regular CAPTCHA. Another problem occurs if the TPR is a hyperlink; in this case, the mouse pointer changes when the user rolls over the TPR. Clicking on the link will load another page in the middle of the CAPTCHA. Even if the TPR is not a hyperlink, a real CAPTCHA is presented as an image and the user should not be able to mark the text of the fake CAPTCHA.

To prevent detection, we cover the iframes with transparent elements. We also disable focusing on elements in the iframe using the Tab key; this is done by setting the *tabIndex* attribute of every iframe to -1.

4. MALICIOUS CAPTCHA FOR MULTIPLE BOOLEAN QUESTIONS

The CAPTCHAs we presented so far trick users into providing the attacker with the answer to an open question, e.g., what is your name. In such cases, the attacker needs to trick the user into disclosing all, or at least most, of the characters in the answer. Since the length of the response is usually unknown, the attacker can generally expose at most a single `answer' per CAPTCHA.

However, as we discuss in this section, the attacker is often interested in answers to Boolean or multiple-choice questions, e.g., to test for speci c history or preferences of the user. In such cases, it is not necessary to load the whole response to the CAPTCHA. Instead, it is possible to load a minimal fragment of the response that will be di erent between the cases. Because usually only a single small fragment is necessary to get the answer, the same CAPTCHA can be used to combine several fragments that answer multiple questions.

We describe two examples for such a CAPTCHA. We begin with Facebook and Google, where the attacker learns whether the victim likes something or follows someone. We continue with more complicated cases, in which the attacker can ask questions about terms that might appear in the search history of a Bing user.

4.1 Facebook and Google: Detect Likes and Followed People

To defend against clickjacking attacks, Google and Facebook do not allow third-party sites to load any of their pages in iframes. However, both Google and Facebook o er a `button' (social plugins), allowing their users to \like" items and to follow other users. Websites are encouraged to embed these buttons in their pages to promote their content via social networks.

In Section 3.5, we showed that it is possible to exploit Facebook social plugins to extract the name of the Facebook user; the same attack works for Google. We now show that it is possible to learn additional information in both Google and Facebook. Speci cally, we focus on *followed people* and *liked items*. Both are considered private data with potential for di erent forms of abuse, and both can be extracted from users of Google and Facebook. We now explain how to detect which items a Facebook user likes, among some set of items, and to similarly detect followed Google+ users.

4.1.1 Detect Facebook Likes

Facebook allows third-party websites to embed Like buttons for arbitrary chosen items. In the standard view, a label with some text comes with the button. This text is changed depending on whether the authenticated user likes the item or not. To detect the like status, it is enough to check the position of *some* letter for one of the options, and to decide, based only on its value. For example, if the rst letter is `Y', then this is the beginning of \You and XXXX others like this" or \You like this". Otherwise, either the `B' of `Be the rst of your friends to like this', or the rst digit of the number of liking users might appear. Because one letter is enough to detect whether a given item is `liked', it is possible to use only one of the multiple frames in the CAPTCHA for each `like' validation. This leaves the other frames for a side-channel of other information, such as the `like' of additional items.

4.1.2 Detect Followed Google+ Users

Google+ allows third-party websites to o er a *Follow* button to visitors, specifying any Google+ user pro le. When a user clicks on the button, Google+ adds the user as a follower of the speci ed pro le. Unlike Facebook, there is no label with changing text as a function of the follow status. Instead, the text and the color of the button itself change. The button has the text \Follow" and is colored in white if the user does not follow the user. Otherwise, either \Following" or the number of Google+ circles that is relevant to the followed user, appear on the button with a blue background.

To di erentiate between the cases, we can use some of the techniques presented by Weinberg et al. [28] to distinguish between hyperlinks marked in di erent colors. For example, it is possible to take parts from the buttons that do not contain text, and to write digits with them over a white background. The digits will not be seen if the button is white like the background; it will be visible only if the user follows the candidate and hence the button is blue. By using an appropriate font, a single digit can even be used to test more than one button; see [28].

4.2 Extracting Bing Search History

Search history exposes sensitive information about the user, such as interests, needs, and actions; see [5]. Several works even discuss the challenge of hiding the search queries of users from the search engines themselves; see [4] and references within.

In this subsection, we show how it is possible to expose parts of the search history of users for the Bing search engine. Like other popular search engines, by default, Bing saves its users' search history. This information is used to personalize the search results and the advertisements, as well as to o er more relevant autocomplete suggestions when the user types in the search box.

For each letter that an authenticated user types into the Bing search box, the Bing client-side script sends a request for personalized autocomplete suggestions for the typed term. Bing dedicates the rst autocomplete suggestions to terms that appear in the search history of the user, such that the typed term is a pre x of them.

Our attack exploits two features of Bing's autocomplete mechanism. First, autocomplete suggestions can be loaded in an iframe. Furthermore, Bing allows *cross-site autocomplete requests*. Namely, websites visited by a Bing user can request personalized autocomplete-suggestions in the name of the authenticated user. Bing does not deploy tokens or any other mechanism to prevent cross-site requests, and responds with personalized autocomplete suggestions. History-based autocomplete suggestions are presented rst, such that suggestions based on newer/more popular searched terms appear higher. Consequently, the last search query, or a very popular search query of the victim, usually appears as the rst autocomplete suggestion for an empty term. This is the case if the user just accesses the search box without typing any letter. An attacker can load this suggestion as the TPR of a malicious CAPTCHA attack.

Similar to the Boolean CAPTCHA presented above for Facebook and Google, the attacker can also ask several Boolean questions about the search history in a single malicious CAPTCHA. Say the attacker wants to test whether or not the victim searched for a term $T = t_1 t_2 \ldots t_n$. As-

Table 2: Detecting whether the victim searched each of four terms (T), by loading small parts of the autocomplete suggestions for pre xes of the terms (T'), and checking which letters were actually loaded in the CAPTCHA.

Term (T)	Prefix (T')	First general suggestion for T'	Letters to search in the CATPCHA
Ecstasy	ecs	ecs tuning	ecsta$\underline{\text{sy}}$
LSD	ls	lsu	ls$\underline{\text{d\ }}$
Cocaine	coca	coca-cola	coca$\underline{\text{in}}$e
Heroin	hero	hero monkey	hero$\underline{\text{in}}$

sume also that there is some value $m <= n$, such that the rst general (not personalized) autocomplete suggestion returned by Bing for the term $T' = t_1t_2...t_m$ (a pre x of T), is di erent from T. If the user has not searched for T, by typing T', the rst autocomplete suggestion will not be T.

Trivially, the attacker could just ask for T' and apply the malicious CAPTCHA techniques on the rst autocomplete suggestion (the TPR). However, in this case, most of the TPR is neither important nor relevant. The beginning of the TPR is expected to be T' and is already known to the attacker. The rest of the letters are also not necessary, because the attacker knows the expected position of each letter, when T is the rst autocomplete suggestion. Hence, she can just pick one or two letters, and load only the parts of the page where they are expected to appear. If the victim lls the CAPTCHA solution with the expected letters, then T was probably suggested, which means that T was searched for by the victim.

Instead of asking about multiple terms and checking whether they match the rst autocomplete suggestion, it is possible to use the malicious CAPTCHA to search for a term in several autocomplete suggestions returned by Bing.

4.2.1 Example: Check Four Terms

Assuming the attacker wants to check whether the victim searched for the following drugs: ecstasy, LSD, cocaine, and heroin. Table 2 shows the relevant T' for each of the terms, as well as letters to use in the CAPTCHA. Figure 5 depicts the given CAPTCHA in three cases: (1) the victim has searched for all four terms, (2) the victim has searched for only two terms, and (3) the victim has not searched for any of them.

5. EVALUATION

We conducted two experiments to evaluate the e ectiveness of the malicious CATPCHA attacks described in the previous sections. We focus on the variable-width (proportional) font CAPTCHA, described in Section 3.3.2, since most TPRs appear in such fonts. Moreover, variable-width CAPTCHA poses greater challenges to the attacker, compared to xed-width and simple CAPTCHAs. We speci cally consider two main aspects:

1. Avoiding arousal of the victim's suspicions.
2. Reproducing the TPR.

The rst experiment evaluated the e ectiveness of anagram CAPTCHA, with and without the advanced variants described in Section 3.4. The second experiment evaluated the use of malicious CAPTCHA to get the answers to multiple Boolean questions, as described in Section 4.

5.1 Obfuscation Techniques Evaluation

In this section we describe an experiment that compares the e ectiveness of the anagram CAPTCHA to that of the camou aged anagram CAPTCHA and to that of the two-step camou aged anagram CAPTCHA. The experiment tests whether users suspect or notice when they type anagrams of their own details, and validates that the attack correctly recovers these details. For the experiment, we chose to extract the name of Facebook users.

5.1.1 Experiment Procedure

At the outset of the experiment, we asked the participants to sign into their Facebook account. Then, we explained that the goal of the experiment was to test whether they noticed being manipulated, where the manipulation may involve a change in a familiar mechanism. We explained that the participants of the experiment were divided into two groups: one that will be manipulated and a control group with no manipulation. In practice, all the participants went through the same procedure. Since the results were so good (lack of detection), there was no need to compare to a control group, which could only produce even better results. Throughout the experiment, the participants could report any suspicious action or indicate that they detected some manipulation.

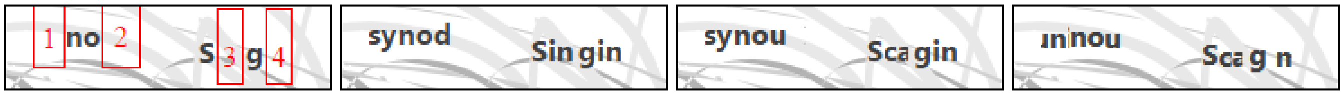
The technical procedure of the experiment included three simple tasks; each involved clicking a link to some Facebook page and answering a question. Before each of the tasks, the participants were asked to solve a CAPTCHA \to prove they are human". We used the basic anagram technique and its two advanced variants mentioned in Section 3.4, from the hardest to detect (two-step camou aged) to the easiest (basic anagram). In the basic anagram CAPTCHA we tried to extract the last name, and in the camou aged CAPTCHA we tried to extract the rst name. In the two-step camou aged CAPTCHA we tried to extract both of them.

In each step of the procedure, below the CAPTCHA or the question, the user was asked to report if there was anything suspicious. Examples of all three CAPTCHAs appear in Figure 4. Notice, for the two-step and the basic anagram CAPTCHAs, the string that comes before the name in the target page \Posting as" may be rearranged together with the TPR.

Ethics. We extracted only information that the user agreed to give us in advance. Namely, the participants of the experiment knew the experiment was on Facebook; they voluntarily and consciously lled in a form with the information that was later stolen. We obtained IRB approval for the experiment.

Participants. A total of 30 students participated in the experiment.

CAPTCHA behavior. In addition to its basic graphical layout and design that was based on the popular reCAPTCHA [27], our CAPTCHA implementation di ered from typical CAPTCHAs in seedl Ca(y)1s:TJ0 .4577810.2.83Td[(ex1(.)-



(a) Template (b) All were searched (c) Only *ecstasy* and *heroin* were searched (d) None were searched

Figure 5: Malicious CAPTCHA to detect whether each of the four terms appears in the search history of the victim (see Table 2). Sub gure (a) is the template of the CAPTCHA; the dummy letters no, S and g are used to detect incorrect input and to separate between the four parts of the CAPTCHA: (1) ecstasy, (2) lsd , (3) cocaine, (4) heroin. The bold letters will appear in the relevant frame of the template if the victim searched for the term.

3. We did not replace the CAPTCHA after incorrect input. Instead, we prompted a message to the user telling her that the CAPTCHA was case sensitive and that) cut letters should be ignored.

All of the above changes only made it easier for the participants to suspect the CAPTCHA. Yet, we found that these changes did not have any e ect on the participants.

5.1.2 Suspicion Evaluation

Once the participants completed the technical procedure of the experiment, we presented three additional Yes/No questions, with the option to add arguments in a text box. The questions gradually gave additional information to the user about the attack that actually happened:

1. Have you noticed anything suspicious?
2. Have you noticed any attempt to trick you into divulging your personal information (such as email address, name, address, etc.)?
3. Have you noticed that you actually inserted your name in some of the CAPTCHA challenges?

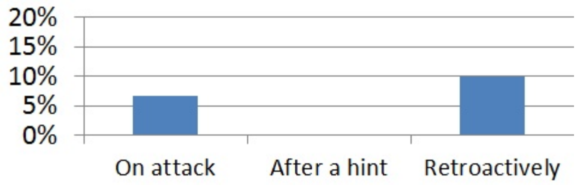
The rst question asks about an attack, the second question describes the attack in general, and the last question tells the user about the attack. Participants with serious concerns were expected to return a positive answer to the rst question. Others, who might have only the slightest doubt, were expected to report in the last question.

Participants that returned a positive answer for the last question, also reported the CAPTCHA in which they noticed the name.

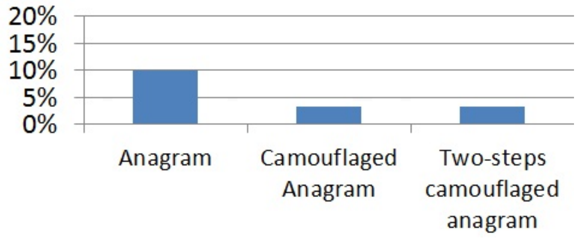
Results. As we expected, the detection rate was low. During the experiments, only two participants reported that they saw their names in one of the CAPTCHAs and became suspicious. Even after we asked the second question and hinted to the participants that the manipulation was about tricking them to disclose details about themselves, no additional users indicated anything suspicious. Once we told the participants what really happened, three of them indicated that they saw fragments of their names. Figure 6(a) depicts these results. Figure 6(b) presents the number of participants who were suspicious of each of the CAPTCHAs. The results indicate that the camou age techniques indeed make it harder to detect the content.

The results of the experiment are even more impressive, considering its environment. In a real website, no alert for upcoming manipulation will be given to the victim. Additionally, we used three CAPTCHAs that encapsulated almost the same information; by itself, this fact could help the victim recognize the attack in the later CAPTCHAs.

CAPTCHA complexity. During the experiment, the participants were asked to write down anything suspicious.



(a) Based on the time it took for suspicion to be aroused



(b) Based on the kind of CAPTCHA that rst aroused suspicion

Figure 6: Suspecting participants (%) based on the time it took for suspicion to be aroused and CAPTCHA type

Almost a third of the users wrote something about the di culty of solving the CAPTCHA, or about cut letters. Yet, all of these reports were in the rst CAPTCHA. Once the users noticed the special instructions of the CAPTCHA (capital letters, ignore cut letters), they solved the other CAPTCHAs quickly. The average number of attempts until the input matched the dummy pattern of the CAPTCHA, decreased from 2.23 in the rst step of the two-step CAPTCHA to 1.2 in the second part, and to 1.33 in the camou aged anagram.

5.1.3 TPR Reproduction Evaluation

We now evaluate how e ective it would be for the attacker to extract the sensitive information from the inputs typed by the participants. Notice, even if the users do their best to ll-in a correct response to the CAPTCHA, there may be mistakes due to accidental typos, or, more often, due to incorrect reading of the text in the CAPTCHA. Some letters may appear more than once, completely or partially. For example, a cut m, might be seen as an r or n. See Section 3.3.2.

We rst describe how we reproduced the TPR and then present the evaluation of the reproduction process.

Reproducing the TPR. The correct text can be reconstructed by taking advantage of the n-gram distribution of the text, by estimating the likelihood of duplication between speci c characters, and by the font used in the browser. In our experiment, we manually recovered the TPR according to a simple algorithm. Namely, we sorted the fragments by their real order and omitted dummy characters. Then we omitted characters that appeared twice, due to the overlapping between fragments. For example, if the input for one fragment ended with r, and the pre x of its next fragment's input was m, then we considered removing the r character. Similarly, we removed thin characters that were likely to actually result from the beginning or the end of other letters, e.g., I and l, as the beginning or end of M or H. During the whole procedure, we used a CAPTCHA simulator that receives a name and shows what the CAPTCHA would look like for a user with this name. We stopped when we got a real name, for which the input of the user matched the text that actually appeared in the CAPTCHA simulator for this name.

Evaluating the reproduction. Because the solutions of the three CAPTCHAs overlap, we asked three volunteers to reproduce the names; each reproduced the solutions of one kind of CAPTCHA. In the two-step CAPTCHA, there was only one mistake of an additional i letter in the middle of an uncommon last name (and no mistake in rst names). In the camou aged CAPTCHAs, all the rst names were recovered successfully. In the basic anagram CAPTCHA, only one last name could not be recovered.

Our results show that it is possible to reproduce the TPRs with a high success rate.

5.2 Evaluating Malicious CAPTCHA for Multiple Boolean Questions

In Section 4, we described variants of malicious CAPTCHA that can be used to answer multiple Boolean questions. In the end of the section, we presented an example that describes how to detect which drugs (if any) out of four options the user searched for in the Bing search engine. We now describe a small study that uses this example to evaluate the e ectiveness of the technique described in Section 4. The experiment was conducted with the participation of 20 student volunteers.

In the experiment, each participant was asked to search for some or none of the four drugs mentioned in Section 4.2.1. After the search, the participant lled in a form with her details and solved the CAPTCHA illustrated in Figure 5. Based on the CAPTCHA solution, we reproduced which of the 16 search combinations was done by each participant, and veri ed the correctness with them. To complete the experiment, we asked the participants to guess how we reproduced their searches.

The results of the experiments were *perfect*; *all* the searches were reproduced perfectly and *none* of the participants suspected the CAPTCHA.

6. DEFENSES

Similar to clickjacking, the malicious CAPTCHA attack relies on loading webpages in iframes; see Section 2.2.

The XFO header prevents the inclusion of speci c web objects, such as iframes, in other sites. Hence, it is also an e ective defense against the malicious CAPTCHA attack. The same holds for `frame-busting' techniques [21, 24], used to prevent the framing of objects by browsers that do not support the XFO header.

However, there are common scenarios where there is strong motivation to allow third-party sites to include a personalized web object within an iframe, such as social media buttons and widgets. In addition, some attacks were shown allowing the circumvention of the XFO header and/or of frame-busting techniques, e.g., [21, 24]; these attacks are equally relevant to the use of these defense mechanisms against malicious CAPTCHA. This motivated other defenses against clickjacking [1, 3, 17]. We therefore evaluated whether these defenses would also protect against the malicious CAPTCHA attack.

Some popular clickjacking-countermeasures do not seem to defend against malicious CAPTCHA. These include defenses based on additional interaction with the user [8], and defenses based on the detection of transparent iframes that are placed on top of other elements [3]. However, it may also be feasible to detect the malicious CAPTCHA attack by extending the detection approach of Balduzzi et al. [3]. Another anti-clickjacking defense that seems to help against the malicious CAPTCHA attack, is to randomize the position of elements [16].

An alternative simple method to prevent malicious-CAPTCHA attacks is to avoid *presentation* of any private information, at least not in a directly visible form amenable to the attack (and to use in CAPTCHA). For example, in many social-network buttons, it may be possible to simply remove the private information currently displayed (e.g., name), optionally exposing it only after some simple action by the user. For example, in the Facebook comment box, the name of the authenticated user may appear only when the user focuses on the text box, or by popping up an annotation on typing. Attackers could try to manipulate the user into focusing on or clicking on elements, but we doubt this can be done e ectively enough to become a signi cant threat.

A longer-term defense may be an extension of the XFO header, allowing websites to include some objects in iframes, but with restrictions to prevent the malicious CAPTCHA attack. In particular, sites may restrict the use of other layers that hide part of the frame or the number of inclusions of the frame in the same webpage. Some of these defenses may even be useful to implement as client-only defenses. For example, the webpage could adopt a default policy of preventing more than several inclusions of the same object or of tiny frames.

7. RELATED WORK

The idea of exploiting web users by making them ll in CAPTCHAs has also been used for legitimate purposes. The ReCAPTCHA project [27] used the text challenge to include images of words that optical character recognition (OCR) software has been unable to read. The solutions of the CAPTCHAs, submitted by millions of users every day, have been used to digitize millions of books and articles.

Our work is not the rst to prompt CAPTCHA for malicious purposes. Instead of using dedicated human resources to manually break CAPTCHA [7], Egele et al. o ered to inject prompted CAPTCHA challenges of one web-service into other web-services, and to take advantage of their users, by forcing them to solve the CAPTCHAs [9].

Weinberg et al. [28] explained how to detect links that were followed by the victims, based on the link's style; this work exploits the fact that the links followed are presented di erently to the users. Similar to our attack, although the attacker can present links to the victim from her website, it is impossible for her to see how they are actually presented by the browser. Weinberg et al. created several CAPTCHAs out of the pixels of these links; hence, by answering the CAPTCHA, the user exposed the color of the pixels. In this way, the attacker learned whether the link was followed or not. Our attacks extract much more information from the victim site itself. However, the techniques presented by Weinberg et al. [28] to combine many Boolean questions in one CAPTCHA, can be used to optimize the multiple-questions in malicious CAPTCHA attacks, as described in Section 4.

Unlike the malicious CAPTCHA attack, which is suitable to expose sensitive private information from many websites, Weinberg et al. focused on speci c Boolean questions: whether or not the victim browsed to a URL with her browser.

8. CONCLUSIONS

We showed a simple and e ective attack that allows attackers to expose private details presented by websites. This is done by exploiting the users themselves as a side-channel, tricking the users into disclosing their own private information. The attack works using all standard browsers, and against some of the most well-known and guarded software-as-a-service web-services, such as Google and Facebook (see more in Table 1).

Similar to other web attacks, defending against this sort of attack is not very di cult, as we explain in Section 6. However, appropriate defenses, especially short term defenses that do not assume new client-side mechanisms, may require web services to slightly modify some mechanisms, e.g., social-network buttons. Such changes may involve a small loss in functionality. It would be interesting to see if and how the industry responds to this challenge: will the small loss in functionality be accepted in order to protect user privacy?

9. ACKNOWLEDGMENTS

This research was supported by a grant from the Ministry of Science and Technology, Israel.

10. REFERENCES

[1] D. Akhawe, W. He, Z. Li, R. Moazzezi, and D. Song. Clickjacking revisited: A perceptual view of UI security. In *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, San Diego, CA, Aug. 2014. USENIX Association.

[2] Alexa. Top Sites. `http://www.alexa.com/topsites`, 2015.

[3] M. Balduzzi, M. Egele, E. Kirda, D. Balzarotti, and C. Kruegel. A solution for the automated detection of clickjacking attacks. In *Proc. of the 5th ASIACCS, ACM Symp. on Information, Computer and Communications Security*, pages 135–144. ACM, 2010.

[4] E. Balsa, C. Troncoso, and C. Diaz. OB-PWS: obfuscation-based private web search. In *Security and Privacy (SP), 2012 IEEE Symposium on*, pages 491–505. IEEE, 2012.

[5] M. Barbaro and T. Zeller. A face is exposed for AOL searcher no. 4417749. *New York Times*, Aug. 2006.

[6] A. Bortz and D. Boneh. Exposing private information by timing web applications. In *Proceedings of the 16th international conference on World Wide Web*, pages 621–628. ACM, 2007.

[7] Brad Stone. Breaking Google CAPTCHAs for Some Extra Cash. `http://bits.blogs.nytimes.com/2008/03/13/breaking-google-captchas-for-3-a-day/?_r=0`, 2008.

[8] Chester Wisniewski. Facebook adds speed bump to slow down likejackers, March 2011. Online at `https://nakedsecurity.sophos.com/2011/03/30/facebook-adds-speed-bump-to-slow-down-likejackers/`.

[9] M. Egele, L. Bilge, E. Kirda, and C. Kruegel. CAPTCHA smuggling: hijacking web browsing sessions to create CAPTCHA farms. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, pages 1865–1870. ACM, 2010.

[10] Eric Lawrence. ClickJacking Defenses. `http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx`, 2009.

[11] Facebook. Social plugins, February 2015. Online at `https://developers.facebook.com/docs/plugins`.

[12] A. J. Ferguson. Fostering e-mail security awareness: The west point carronade. *EDUCASE Quarterly*, 2005.

[13] N. Gelernter and A. Herzberg. Cross-site search attacks. In *Proceedings of the 22nd ACM Conference on Computer and Communications Security*, CCS '15, pages 1394–1405, 2015.

[14] R. Hansen and J. Grossman. Clickjacking. *Sec Theory, Internet Security*, 2008.

[15] A. Herzberg and R. Margulies. Forcing Johnny to login safely. *Journal of Computer Security*, 21(2):393–424, 2013.

[16] B. Hill. Adaptive user interface randomization as an anti-clickjacking strategy, May 2012. `http://www.thesecuritypractice.com/the_security_practice/papers/adaptiveuserinterfacerandomization.pdf`.

[17] L.-S. Huang, A. Moshchuk, H. J. Wang, S. Schecter, and C. Jackson. Clickjacking: Attacks and defenses. In *USENIX Security Symposium*, pages 413–428, 2012.

[18] D. Irani, M. Balduzzi, D. Balzarotti, E. Kirda, and C. Pu. Reverse social engineering attacks in online social networks. In *Detection of intrusions and malware, and vulnerability assessment*, pages 55–74. Springer, 2011.

[19] T. N. Jagatic, N. A. Johnson, M. Jakobsson, and F. Menczer. Social phishing. *Communications of the ACM*, 50(10):94–100, 2007.

[20] Jeremiah Grossman. I Know What Websites You Are Logged-In To (Login-Detection via CSRF). `http://blog.whitehatsec.com/`, 2009.

[21] S. Lekies and M. Heiderich. On the fragility and limitations of current browser-provided clickjacking protection schemes. In E. Bursztein and T. Dullien, editors, *WOOT*, pages 53–63. USENIX Association, 2012.

[22] Mozilla Developer Network. Same-Origin Policy, 2014. `https://developer.mozilla.org/en-US/docs/Web/Security/Same-origin_policy`.

[23] Mozilla Developer Network. Using the application cache, 2015. `https://developer.mozilla.org/en-US/docs/Web/HTML/Using_the_application_cache`.

[24] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In *in IEEE Oakland Web 2.0 Security and Privacy (W2SP 2010)*, pages 1–13, 2010.

[25] The Open Web Application Security Project. Cross-Site Request Forgery. `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)`, 2010.

[26] T. Van Goethem, W. Joosen, and N. Nikiforakis. The clock is still ticking: Timing attacks in the modern web. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1382–1393. ACM, 2015.

[27] L. Von Ahn, B. Maurer, C. McMillen, D. Abraham, and M. Blum. reC823.T Ha-bwaed, Cer
Rcongn v(a)-354(W)89(eb)-383(e(c)1(urit*S*29ey d)-353(M)1ewaurtes.
258 5)68M, 020
8 . Wien M..(n,) 54PW. . atyhamn, and
c(so)1(n.) 54(.) 5 ustillnowo v(sitdt) 5 laestmm
kingowsinghist oydneraation and
Sauttdgk2ndnPivcty (S 201 IIE.
Smcoume n