Autocomplete Injection Attack

Nethanel Gelernter[1,2,3](✉) and Amir Herzberg[2,3]

[1] Cyberpion, Givat Shmuel, Israel
nethanel.gelernter@gmail.com
[2] College of Management Academic Studies, Rishon LeZion, Israel
[3] Bar Ilan University, Ramat Gan, Israel

Abstract. Autocomplete, a well-known feature in popular search engines, offers suggestions for search terms before the user has even completed typing their query. We present the *autocomplete injection attack* and its potential exploits. In this attack, a cross-site attacker injects terms into the autocomplete suggestions offered by a web-service to a victim user. The most popular web search engines are vulnerable to the attack, as well as other websites.

Autocomplete injection can be exploited in multiple ways, including *phishing, framing, illegitimate content-promotion* and sometimes *persistent cross-site scripting attacks*. We evaluated the effectiveness of the attack with several experiments. Our results show the potential impact of the autocomplete injection attacks.

Keywords: Web-security · Phishing · Cross-site attacks · Usable security · Autocomplete injection attack · Cross-site framing · Blackhat SEO · Cross site scripting · Persistent XSS · CSRF

1 Introduction

Web-services invest considerable efforts to improve their user experience. More specifically, services are often *personalized* using information collected about each user, including the history of previous interactions. *Autocomplete mechanisms* are one of the personalization methods most widely-used by web-services to ease the entry of search terms. As the user types the first few letters of a query, the autocomplete mechanism offers several suggestions for the complete query. The user can either avoid typing the rest of the term by choosing one of the suggestions or type additional letters, which will prompt updated suggestions. Autocomplete suggestions allow users to choose long terms while decreasing the number of keystrokes [3,27]. This serves to improve the user experience, especially for users with disabilities (for whom it was originally designed).

We show that the autocomplete mechanism can also be *abused* by a rogue website visited by the user, allowing multiple attacks on those who have an active 'session' in one of several popular websites. Based on this new type of 'cross-site' attack, we tested for - and found - autocomplete vulnerabilities in five sites: the

I. Askoxylakis et al. (Eds.): ESORICS 2016, Part II, LNCS 9879, pp. 512–530, 2016.
DOI: 10.1007/978-3-319-45741-3_26

three most-popular search-engines: *Google, Yahoo!* and *Bing*, and the three most popular websites (Google, Facebook and Youtube) [2].

This paper shows how it is possible to manipulate the autocomplete suggestions, and demonstrates and evaluates this possibility on Google, Yahoo! and Bing. Our results show that it is possible to control the first autocomplete suggestion that will be offered by these websites to the victim, for almost every prefix she types. We refer to this manipulation as the *autocomplete injection attack*.

Autocomplete injections can be exploited in different ways to benefit the attacker and harm the user. We present four ways in which autocomplete injections can be exploited by attackers: *phishing, framing, illegitimate content-promotion*, and *persistent cross-site scripting attacks*. We discuss each of these exploits in a dedicated section, and briefly introduce them below.

Phishing Attacks (Sect. 3). In phishing attacks, the victim is tricked into visiting a malicious 'imposter' website that mimics a legitimate site [12]. Tricking the user into visiting the imposter site is a critical part of the attack. In most phishing attacks, this is done by the user clicking on a link from a phishing email, website, or ad [23]. However, this 'direct' approach has disadvantages, most notably, the victim accesses the imposter page as a result of an external event (e.g., email or ad), and not as a result of the user's own agenda; this may make the user more alert upon visiting the phishing site, especially when the user is aware of the threat of phishing attacks. The autocomplete injection attack allows launching sophisticated phishing attack; that attack tricks the user into visiting a malicious page by clicking on a result in her favorite search engine.

Illegitimate 'Black-Hat' Content Promotion (Sect. 4). The autocomplete injection attack can be used to promote content. So far, the methods to promote content include the use of ads and public-relations methods, or Search-Engine Optimization (SEO) techniques, including illegitimate *blackhat-SEO* methods. The goal is to cause search engines to return the website as one of their top results, increasing the likelihood of access by users. Content promotion via autocomplete injection is significantly easier, and complementary: the promotion occurs via the autocomplete suggestions for related terms. Note that autocomplete-injection-based content promotion does not require users to search for a particular term, and can work with many or most search queries. This is somewhat similar to the *search poisoning* blackhat-SEO technique [21].

Cross-site Framing Attacks (Sect. 5). The personalization of autocomplete features causes different people to get different suggestions. This fact is known to many web-users. An attacker that is able to control these autocomplete suggestions can mislead the environment of the user, such as spouse, family, and colleagues, into believing the user was surfing sinister content or to draw incorrect conclusions about her interests. Such attacks are a variant of *cross-site framing attacks* [8]. For example, it is possible to plant autocomplete suggestions that will give indications that a married user is interested in dating services, and even in a particular type of dating services (e.g., with specific sexual orientation). In this paper, we evaluated even more severe framing attacks, in which the attacker plants pedophile-related terms in the autocomplete suggestions of her victims.

Persistent Cross-Site Scripting (Sect. 6). The autocomplete injection attack allows a rogue website, to manipulate a popular web-service into sending users attacker-controlled autocomplete suggestions. If the attacker can include a script as part of the autocomplete string, this may allow a *Persistent Cross-Site Scripting attack* [16]. We found that Yahoo! is vulnerable to such an attack and demonstrate how an attacker can manipulate the autocomplete suggestions of Yahoo! such that for every typed letter, the malicious script of the attacker will run.

1.1 Contributions

Our main contribution is the introduction of the autocomplete injection attack. We demonstrate how the attack can be used for four different purposes: phishing, framing, illegitimate content-promotion, and persistent cross-site scripting attacks. We evaluated the applicability of these attacks on highly popular sites (Google, Yahoo!, Bing, Facebook and Youtube), and found that all are vulnerable, allowing phishing, framing, and illegitimate content-promotion. We further found that the autocomplete-injection attack on Yahoo! allows persistent cross-site scripting.

We hope that the publication of this paper will urge websites, including the very popular ones we tested, to protect their users against this threat. We informed all websites of the vulnerability.

Ethics. This research involved several IRB-approved experiments to measure the effectiveness of different autocomplete attacks on users. Ethics was a major issue in the planning of all these experiments, which were designed to avoid causing damage to users and web-services. We describe the relevant ethical considerations for every experiment. We informed all the websites about this vulnerability, giving them sufficient time to address it. Indeed, Yahoo! blocked the cross-site scripting vulnerability posed by the autocomplete injection attack.

1.2 Related Work

Phishing. The idea of tempting the victim into naturally contacting the attacker was presented by Irani et al. in the context of social networks [14]. In their reverse social engineering attack, the attacker tricks the user into creating the initial contact with the attacker, without using classical spear phishing techniques [23] that are often detected by users with some security background.

Illegitimate Content Promotion and Search Engine Optimizations (SEO). Current research on illegitimate content promotion focuses on 'blackhat-SEO' techniques [6,21]. Xing et al. [29] presented an SEO attack based on polluting the search history of users, causing the search engine to offer them different results. We are not aware of previous proposals to promote content or websites by abusing the autocomplete mechanism.

Cross-Site Framing Attacks. Framing attacks in the cross-site adversary model were presented recently [8]. However, unlike most attacks noted [8], injected autocomplete suggestions are likely to be noticed by individuals who can see the victim's display, such as family and co-employees. This can cause severe damage without requiring additional intervention by the attacker (e.g., contacting police and reporting about someone).

2 Adversary Model and Autocomplete-Injection Attack

This section briefly explains the adversary model in this work (Sect. 2.1) and reviews the technical aspects of injecting autocomplete suggestions into websites (Sect. 2.2). The applications of these manipulations are discussed and analyzed in the following sections.

2.1 The Cross-Site Adversary Model

Cross-site attacks require only a modest capability from the attacker: controlling a 'rogue' web-page visited by the victim user. A cross-site attack sends the victim user a web-page, often containing a script that causes the victim's browser to issue requests to some *target* web-service; such requests are often referred to as *cross-site requests*. Cross-site requests are essential to the use of the web and generally used for legitimate purposes. In particular, search-engines and other popular sites use cross-site requests to allow third-party sites to perform search queries. Cross-site requests are also used by well-known web attacks, such as cross-site scripting (XSS) [16], cross-site request forgery (CSRF) [28], cross-site search [9] and clickjacking [25].

Luring random or specific victims into visiting a rogue website is considered an easy task, e.g., using phishing emails, ads, and social-engineering techniques [7,14,15].

To manipulate the autocomplete suggestions offered by a target website, the victim who visits the rogue website must be logged into that website from the same browser. Several well-known techniques allow a cross-site attacker to efficiently detect such cross-site login [4,9,17].

2.2 Injecting Autocomplete Suggestions

A website allows cross-site requests, if it serves requests that were sent from another domain as though they were issued (legitimately) by the client currently authenticated to the website. Many web-services and search engines allow cross-site search requests (queries). Namely, websites can send search requests to one of the popular search engines to which the current user (victim) is logged in; the search engines will treat this search request as if it was sent by the victim.

Search engines maintain logs of the searches done by each of their users. These logs are used to personalize services given to the users by the search engines. One use of the search history logs is to offer personalized autocomplete suggestions. Under the assumption that users tend to search for the same terms again and again, search engines offer terms from the search history log as autocomplete suggestions. We denote such autocomplete suggestions as *history-based* autocomplete suggestions. History-based autocomplete suggestions usually appear first, before *general* autocomplete suggestions, which are based on local and global trends.

The autocomplete injection attack exploits the ability to add entries to the search history logs by sending cross-site search request and the fact that websites present history-based autocomplete suggestions.

The attacker sends a cross-site search request with a query and causes the search engine to add that query to the history log. Later, when the user types a prefix of the injected query, she will see the query as an autocomplete suggestion.

2.3 Autocomplete Injection in Popular Websites

Our work focuses on search engines, and in particular on Google, Yahoo! and Bing, the most popular search engines in the US [2]. The same techniques can be applied to other websites such as Facebook and YouTube.

Figure 1 illustrates the way autocomplete suggestions are presented to the user by the three search engines. It is possible to see the differences between the search engines in both the number of history-based autocomplete suggestions and whether history-based suggestions visually differ from the general autocomplete suggestions.

Other differences between the search engines relate to content filtering and the number of cross-site requests required to plant a term. Google is the only search engine that filters out content related to pornography, violence, hate

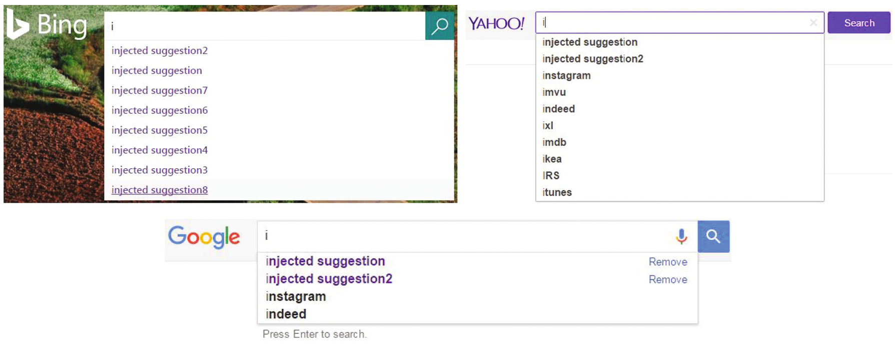


Fig. 1. Autocomplete suggestions as presented by the Google, Yahoo! and Bing search engines in US. Injected suggestions appear as history-based autocomplete suggestions.

Table 1. Differences between the three most popular search engines [2] with regarding to history-based autocomplete suggestions as presented to their users

	Maximal number	Colored differently	Filtering	Number of searches to be added
Google	2	$\surd$	$\surd$	1–3
Yahoo!	2	$\surd$	X	1
Bing	8	X	X	1

speech, and illegal and dangerous objects [10]. This fact is relevant mainly for the framing attacks using autocomplete injection and discussed in Sect. 5.1. Google also differs from the other search engines in the number of cross-site search requests required for a term to appear as an autocomplete suggestion. We created a webpage that sent cross-site search requests for a list of 14 terms. These terms included meaningful and meaningless terms, and other types of terms that are being injected during the attacks described in this paper. The page sent only a single request for each of the terms, and we checked whether the term appears as autocomplete suggestion or not. In both Yahoo! and Bing, all the autocomplete suggestions were presented immediately, but in Google some of them did not appear. Hence, we repeated the test on 10 active Google accounts when 2 and 3 requests are sent. After sending 2 cross-site search requests for each term, the term appeared in 95 % of the cases. *All* the terms appeared after sending 3 cross-site search requests. Table 1 summarizes the differences between the search engines.

3 Phishing

The ability to manipulate the autocomplete suggestions seems ideal for phishing attacks. Theoretically, the attacker simply plants autocomplete suggestions that mislead the search engine to offer the phishing website instead of the original one. A user that relies on the injected autocomplete suggestion will get the phishing website in her search results.

Although the idea seems simple, in practice it is not easy to manipulate the search engine to offer incorrect results for terms that are related to popular websites. In this section we describe the challenge of the attacker and offer two techniques that can be used to launch a successful phishing attack using autocomplete injection. At the end of the section, we present the results of an experiment we conducted to evaluate these techniques.

3.1 The Challenge

When users search a popular page they usually type the name of the website or the name of the organization associated with the website. Even if the user incorrectly spells a popular term, the search engines usually offer results that are based on the correctly-spelled term. Similarly, combining terms that are strongly related to a (popular) website with the name of another website will yield search results that are related to the original website. For example, if an attacker concatenates the name of a phishing website to the name of some bank in an autocomplete suggestion and the user searches this suggestion, the search engines will return entries from the bank website and the phishing website will not appear in the first entries.

This poses a challenge to the attacker; the attacker has to find autocomplete suggestions that will satisfy the following conditions:

1. Relevant for the search so the user will select them.
2. Yield search results without the real website appearing at the top.
3. Yield search results with the phishing website at the top of the list.

3.2 Phishing Autocomplete Suggestions

We describe two techniques to create autocomplete suggestions that satisfy the three requirements of the phishing challenge. The first uses advanced search operators and the second relies on homographs.

Advanced Search Operators. Although most of the users rely on the simple search, search engines also support advanced search operators. The attacker can use some of these operators to abuse the search results.

The first operator is the *not* operator, usually denoted by the minus sign (-). Attaching the *not* operator to a term means searching for results that do not contain that term. The attacker can abuse the *not* operator in terms that contain more than a single word. For example, for a term such as *bank Somebank* that can be searched also as *bank-Somebank*, the attacker can inject the autocomplete suggestion *bank -Somebank <bait>*. This will cause a search for results *without* the name of the bank, *Somebank*, and *with* the terms *bank* and *<bait>*. Here, *<bait>* is a search-term that is bound to cause the phishing-website (e.g., its domain name *phishingsite.com*) to be the top result.

Another operator that can be used for this purpose is the *site*/*inurl* operator. This operator specifies the site/url of the results. By concatenating this operator after relevant search terms to specify that the results must come from the phishing website, the attacker ensures that the results will be only from the phishing website.

Homographic Autocomplete Attacks. Another direction is to use homographs. A homograph is one of two or more characters, or sequences of characters, with shapes that either appear identical or cannot be differentiated by quick visual inspection. Homographs are usually used to deceive users in classical phishing attacks [1,13,22]. For example: replacing the English letter *o* (code 006 F in Unicode) in "Bank *o*f America" with the Cyrillic small letter "o" (code 043E in Unicode). A search for this simple homograph in Google brings the real result for Bank of America only in the eighth entry. By replacing also the two appearances of the English letter *a* in the word "america" (autocomplete suggestions always appear in small letters) with the Cyrillic letter "a" (code 0430 in Unicode), the attacker gets a homograph that will yield search results without the real website of Bank of America.

Once the attacker has a homograph that yields search results without the real website, the attacker needs to create a phishing website that will appear in the first results for that homograph. This is not considered a hard task, because the homograph used by the attacker is not a common search term.

When choosing the homograph, it is preferable to replace several characters. Otherwise, the search engines might refer to the term as a typo and, and due to the similarity to a popular real term, will present the results for that term. However, to increase the exposure of the autocomplete suggestion, it is better to replace characters toward the end of the term, such that the homograph and the real term share a prefix that is as long as possible.

3.3 Evaluation

To evaluate the autocomplete injection attack for phishing purposes, we designed a one-minute usability experiment. The experiment evaluated both the techniques described in Sect. 3.2.

Experiment 1: Bank Phishing

Goal: Check whether users will choose injected autocomplete suggestions that will lead to phishy search results for each of the techniques described in Sect. 3.2. Also check whether they will follow the phishy search results to the phishing website.

Methodology and Ethics: The experiment was carried out with 100 volunteer undergraduate students in a security course and with employees working in a security firm. The participants agreed to participate and did the experiment on their computers. To avoid 'contaminating' their real accounts, the experiment used a dedicated search page, which used the autocomplete suggestions and search results of Bing, by presenting Bing's results in an iframe[1]. We added to this page the "injected" autocomplete suggestions.

To analyze the phishing results, we created a page with search results for the phishing terms and loaded this page as the search results when the user clicked on the phishing autocomplete suggestions. This was the alternative for buying a phishing domain, creating a real phishing website and promoting it in search engines.

Process: We chose one of the three largest banks in our country, and instructed the users to find its website using our search engine and to open the website in a new window and then to press on a button to finish the experiment. For each participant, we only saved whether the person used the phishing or legitimate

[1] Yahoo! and Google prevent presentation of search results in iframes.

autocomplete suggestions or none at all, and whether they clicked on the phishing website. The experiment was completed once the user clicked on the phishing website or when she opened the real website in a new window and clicked on the finish button.

Phishing Details: For the phishing website, we chose a name that is similar to the name of a real bank. For half the participants we used the advanced search operators as described in Sect. 3.2 (the *AO* group). For the other participants we used a homograph in which we replaced two English letters with two Cyrillic letters as described in Sect. 3.2 (the *HG* group). The phishy autocomplete suggestions for both the AO and HG groups yielded a page where the first search result points to the phishing website. However, for each autocomplete suggestion s', a homograph of s, in the HG group, this page also contained a question that appears by Bing for such searches: "Do you mean s"? We found that for these particular autocomplete suggestions, both Yahoo! and Bing prompt this question but Google does not. We also asked the users to report any unusual or suspicious behavior they noticed.

Results: We observed that using homographs is much more effective than using advanced search operators. Users from the AO group almost completely avoided the phishing autocomplete results; only two of them searched for the injected phishing autocomplete suggestions, and both users also followed the results to the phishing website. The explanation we got for the lack of use in the autocomplete suggestion was that the phishy suggestions that rely on advanced search operators did not reflect the exact search the users planned to submit. In the HG group, 26 % of the users used the homograph autocomplete suggestion. However, only 20 % clicked on the phishing website. The other 6 % simply clicked on the question raised by the search engine ("Do you mean ...?") and got new search results without any phishing website. Figure 2 depicts the results.

Our most important observation is that *no user reported* a suspected phishing attack. Although users probably noticed the unusual autocomplete suggestions, mainly in the AO group, none of them linked this anomaly with a phishing attack. This means the attacker can launch the attack on a large scale without being

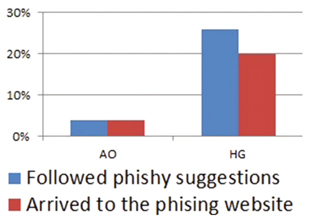


Fig. 2. Users who clicked on injected phishy autocomplete suggestions and users that followed the phishy suggestions and clicked on the phishing website in the search results. The results separate users who were tricked using advanced search operators (AO) and users who were tricked using homographs (HG).

worried about detection. Moreover, the autocomplete-injection attack allows an attacker to exploit a random visit to his rogue website, and manipulate the user ('Alice') into accessing the phishing website later on, when Alice *initiates* a visit to her bank. These are significant advantages compared to classical spear phishing attacks [23]. Specifically, it is far more likely that Alice will 'fall for' a phishing site, when Alice initiates the visit to the bank [14].

Although only 20 % of the users were phished in the experiment, the results indicate that autocomplete injection can be used for effective phishing attacks. The autocomplete injection attack is effective only against users who rely on autocomplete suggestions. When it comes to users who relied on autocomplete suggestions, the homograph variant of the attack achieved a 76 % success rate.

4 Illegitimate (Black-Hat) Content-Promotion

Companies and organizations spend large amounts of money to promote products, slogans, and other content, using advertising and PR campaigns. Hacktivists and 'black-hat organizations', may use illegitimate mechanisms to promote content; these tend to be more effective or less expensive. For example, these may be used to promote illegal or illegitimate content, which may be banned by legitimate advertising and PR providers. In this section we explore the potential abuse of the autocomplete injection attack to perform *illegitimate content promotion*, delivering messages and slogans to website visitors. The content may include malicious content such as the promotion of malware websites, or be part of phishing campaigns, complementing the mechanisms described in Sect. 3.

The attack allows a website, visited by a user, to add a desired string to the autocomplete suggestions offered by popular search engines/sites. This string could be a slogan or other text used to 'promote' some product or idea. It may also be a 'negative text', e.g., discrediting a competitor. Because such goals could be attractive to many attackers, the ease with which a website can launch the autocomplete injection attack makes it a real risk.

This section evaluates the promotion of a slogan using the autocomplete injection attack. Slogans have considerable marketing value, and organizations invest considerable effort and funds to promote and advertise them. Autocomplete-injection even allows the slogan/phrase to appear for users typing relevant terms, for highly-effective targeted advertising. A company can inject slogans and advertisements in the autocomplete suggestions of its website visitors and influence their searches in external search engines. For example, assume a company wants to promote the Doritos® chips brand. The autocomplete-injection can display a relevant slogan, such as the fabricated slogan 'America likes Doritos' (as we used in Experiment 2) arbitrarily or when the user types relevant words and phrases, e.g., *snack*.

Experiment 2: Promoting a Slogan

Goal: Validate that injected autocomplete terms are noticed by users. Evaluate their impact on users and their potential for exposing users to a slogan using the fabricated slogan 'America likes Doritos'.

Methodology and Ethics: The experiment was carried out with 95 volunteer undergraduate students who signed a consent form, and used their own computers to simulate reality. To avoid (unintentional) bias by participants and/or staff, the experiment was 'double blinded', i.e., users were assigned randomly to one of three sets (no injection, or one of two injection modes described below), without awareness of either user or staff. To avoid injecting suggestions to the accounts of the users, we used a search page we built as described in Experiment 1.

Process: Users were instructed to use our search form to send the search queries of their choice. Users accessed a webpage with instructions to run searches during five minutes using the provided search form. Each user was randomly (and blindly) assigned to one of three groups: *None, Letters* and *Terms*. For *Letters* users, we inserted autocomplete strings consisting of each letter in the alphabet concatenated with the slogan, and also each letter repeated (e.g., *aa*). For *Terms* users, for each alphabet letter we chose a popular search term T beginning with this letter, and concatenated the slogan to T in two ways: "T : slogan" and "T - slogan". For *None* users, nothing was injected[2]. After five minutes of search, we redirected the user to a form with a single question: *What does America like?*. There were five possible answers: (1) Fries. (2) Waffles. (3) Kinder. (4) Doritos. (5) I don't know.

Results: As can be seen in Fig. 3, users in both of the 'autocomplete-injection' groups were far more likely to select the answer corresponding to the slogan, compared to users in the control group ('none'). While the number of users is not sufficient for this study to be conclusive, it gives a good indication that users notice autocomplete injections and are influenced by them. These results also show that people notice injected autocomplete suggestions, even in short free searches. This means autocomplete-injection may be effective as a way to 'frame' an innocent victim user, by presenting suspect autocomplete suggestions in the victim's computer; see next section.

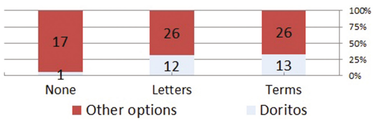


Fig. 3. Participants' answers to the question "What does America like?" in each of the modes

[2] The probability of getting each of the *Letter* and the *Terms* modes was twice the probability of getting the *None* mode.

5 Framing Attacks

This section evaluates the effectiveness of framing attack using autocomplete injection and the damage that such an attack can cause. By planting autocomplete suggestions, the attacker can create a false impression about the victim. For example, autocomplete suggestions that are related to dating sites indicate that the user is interested in finding dates. In cases where the user already has a spouse, such autocomplete suggestions might create relationship issues if observed by someone else.

We investigate the likelihood that a casual user will notice *and* take action, when faced with (fake) autocomplete phrases; these phrases were chosen to create an impression of searches for pedophile contents. This is significant, since autocomplete phrases are automatically offered to any user of the browser. Even casual users, using somebody else's computer and with no intent to snoop, may be exposed to them. This is in contrast to other methods of computer framing [8], which are generally unlikely to be viewed by a casual user. For example, Google requires users to re-authenticate before presenting their search history (and only a snoopy visitor would even try to look up the search history).

The findings of Experiment 2 show that users often notice the injected autocomplete suggestions. However, the fact that users notice autocomplete phrases does not necessarily imply that this can be *noticeable* framing evidence. First of all, users may not deduce from observing the autocomplete phrases that there are implications regarding the user of the computer. More significantly, even if they do, they may not feel the need or confidence to report this. This *bystander* phenomenon, where eyewitnesses fail to report crimes, has been reported and studied in many social-science studies and experiments [11,19].

We therefore conducted Experiment 3 to evaluate the potential abuse of the autocomplete mechanism, as noticeable framing (false) evidence. We focused on autocomplete phrases that seem to indicate searches for pedophilia sites and related activities. Section 5.1 describes how to circumvent Google filtering for inappropriate autocomplete suggestions.

Experiment 3: Bystander and Pedophilia-Autocomplete

Ethical Restrictions and Pilot Experiment: This experiment involved ethical challenges. Clearly, it would not be ethical to inject such phrases into the computers of subjects. In fact, the first author performed such an experiment on his own Google account, by authenticating to the framed account in his home PC, which is also used by his spouse. It took one day until the author was confronted with an upset spouse and had to explain and show that this was just an experiment. For this reason, all further experiments were performed in our lab. Another challenge is the fact that users must see the framing autocomplete suggestions in a natural way and hence must not know the real purpose of the experiment. We explained to the users what we expected them to do and asked for their consent; we only disclosed the real purpose of the experiment toward the very end. Participants were paid and allowed to leave the experiment at any given time without forfeiting the payment. At the end of the experiment, we explained the real purpose of the experiment and repeated our request for their permission to use the collected information.

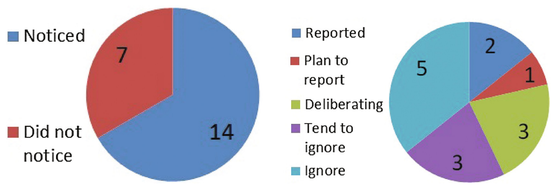
Experiment Design: To achieve the most realistic and reliable results possible, the experiment was designed to emulate a typical workplace situation, searching for terms using Google. After signing consent forms, participants were asked to run 25 web-searches on a computer, supposedly as part of a user experience study. At the outset of the experiment, the computer dedicated to the experiment was found inoperative. This was an excuse to have the participants use a laptop belonging to a contractor, Vic, who worked in the lab and was currently away. Essentially, Vic represents the victim, 'framed' to be suspected of pedophilia. We explained that Vic was away, so we would use his laptop to substitute for the regular experiment computer. We made a phone call to Vic, but he did not respond, so we used the laptop anyway. Supposedly this was the approved, standard practice in our lab.

During the searches, users could see that Vic was 'logged on' to his Google account. In reality, 'Vic's' laptop was configured for the experiment, equipped with a video (camera) and a screen-recording application. It was also injected with pedophilia-related autocomplete phrases. Since Google filters most pedophilia terms, we used homographic variants or 'typos' such as *childp ornography* (see Subsect. 5.1), which we had no difficulty injecting.

Participants were instructed to search for 25 phrases using Google; the phrases were played from an audio file. The phrases included questions like "what is my IP" and "how to find the median", websites like Youtube and Facebook, movies-related search terms, and more. Half of the search phrases had a common prefix with some of the framing autocomplete phrases; we also had many benign autocomplete phrases, to avoid over-visibility.

After searching for the 25 audio-played phrases, participants were asked to run arbitrary searches for 2 more minutes. Participants answered a few statistical questions and were given a paper and envelop for anonymous feedback to be read only by the professor responsible for the lab. This was selected as a comfortable mechanism for them to raise any concerns. To make sure participants did not plan to raise their concerns before leaving, we paid them at this point and allowed them to leave, then asked them to return for two more questions, the first being: *Did you notice any bothersome thing during the experiment?*. If the answer was yes, we asked the participant to write it down. Finally, we explained the real goals of the experiment to the participants, and asked their discretion and their agreement for us to use their results (all agreed). We also asked whether the participants saw any pedophile-related phrases. Participants who reported noticing were asked whether they reported it, and if not, why.

Participants: We recruited 25 participants, all students (ages: 18–38), via ads. We did not include computer science students, since they were expected to have a higher awareness of the autocomplete mechanism. Four students were disqualified after not meeting our minimal threshold of proficiency in English. The complexity of the experiment process prevented us from conducting the experiment on a larger group of participants. Due to the experiment's design, we had to perform the experiment on each participant separately, such that no single participant could observe the experiment of another participant.



(a) Did you notice pedophile-related autocomplete suggestions?

(b) Reporting vs. ignoring the framing attack.

Fig. 4. Framing experiment results

Results and Analysis (see Fig. 4): As shown in Fig. 4(a), two-thirds (14) of the 21 qualified participants, reported that they noticed the pedophilia content. Five of them reported this only when asked explicitly about signs of pedophilia, in the very last question. Even without these 5, 9 (43 %) reported pedophilia content without being asked about it specifically.

We did not have the resources to run a control group without injection, and the number of participants is not sufficient to draw significant conclusions. However, this seems to provide testimony further to Experiment 2, that users notice and are impacted by the autocomplete phrases. Note that in Experiment 2, only about a third of the responders selected the injected slogan (see Fig. 3), possibly because in Experiment 2, the participants were asked to search for whatever they want; they might not have searched for terms with a long prefix of an injected autocomplete suggestion.

Out of the 14 participants that acknowledged noticing pedophilia content, 2 (14 %) used the feedback form to write elaborate complaints to the professor. They explained the pedophilia-phrases and their implications, and demanded immediate and conclusive action, offering to contact authorities themselves otherwise. Of the remaining 14 that reported noticing pedophilia content, 4 reported that they planned to report or considered reporting. Although it may seem that only a small percentage of participants reported, or even planned/considered reporting, the percentage is, in fact, surprisingly high, compared to the 15–25 % reported in most previous studies of the bystander effect, e.g., [11,19].

Our analysis of the recorded behavior of the participants and correlated screen contents offers insight into the reasons seven participants failed to notice the framing pieces of evidence. Apparently the reasons are technical and not very relevant to the use of autocomplete injections and red-flag framing evidence. For example, one of them typed very quickly, such that autocomplete suggestions were almost irrelevant for him (e.g., due to their latency). Most of the others could be seen in the video looking primarily at the keyboard and/or to be on the verge of disqualified for poor English.

5.1 Circumventing Autocomplete Excluded-Phrases

Because Google is the most popular search engine, we used a framed Google account to evaluate framing attacks (Experiment 3). Our focus on injecting pedophilia-related phrases raised a challenge. Unlike the other two search engines, Yahoo! and Bing, Google filters pedophilia-related and certain other ‘problematic’ terms, like pornography, violence, hate speech, illegal and dangerous objects, and terms that are frequently used to find content that violates copyright [10]. Therefore, an attacker who wishes to use Google for framing must circumvent this filtering.

In spite of Google’s filters, it is still possible to inject search queries related to these subjects using simple homographs (see Sect. 3.2). We were able to inject inappropriate autocomplete suggestions using basic homographs, for example: *pornography*, where the first *o* is the Cyrillic small letter “o” (code 043E in Unicode). Another direction is to use minor typos, e.g., *childp ornography*, or adding, dropping, or duplicating letters.

6 Stored Cross-Site Scripting (XSS)

The injection of data controlled by an attacker to a website might result in a cross-site scripting (XSS) attack [16]. The same origin policy (SOP) [24] restricts websites from running scripts that will affect websites with a different origin. The XSS attack circumvents SOP by injecting a malicious script into the attacked page. Once the malicious script is running from the attacked page, it is allowed to access pages and to perform actions in the domain of the attacked page[3].

XSS attacks can be classified into three categories: DOM-based, reflected, and stored attacks. In DOM-based XSS [20], the malicious code is injected in the client side. Stored (persistent) and reflected (non-persistent) are two types of XSS attack in which the server itself returns a page with malicious script. In reflected XSS, the malicious script is reflected from the request, usually from the URL, and hence, it usually requires accessing a malicious link. In stored XSS, the attacker injects the malicious script to the server, and the server embeds this script in some HTTP responses it returns. Among the three types of XSS, stored XSS is considered the most severe. It is harder to detect by browsers, and it requires minimal interaction from the user.

The autocomplete suggestions injected by the attacker are stored by the server and are used later when the user is typing search queries. Therefore, if the attacker injects an autocomplete suggestion that contains malicious script and the server does not filter or sanitize them, the website is vulnerable to a stored XSS attack. The malicious script will be triggered when the user types some prefix of the autocomplete suggestion in which the malicious script was included.

To launch an effective XSS attack, the attacker will want to increase the probability that the malicious script will be run and decrease the user interaction required for the attack. The autocomplete injection allows an attacker to plant the malicious script in a manner that will trigger it for any letter typed in the search box. For every possible character, the attacker can plant an autocomplete

[3] Countermeasures like content security policy (CSP) [26] can mitigate XSS attacks, but are beyond the scope of this paper.

suggestion that is the concatenation of the character and the malicious script. In this way, the website will present the victim with an autocomplete suggestion that will trigger the malicious script execution upon typing every single letter in the search box.

Among the five web-services we tested, we found that only Yahoo! is vulnerable to the autocomplete injection XSS attack. We injected a script into Yahoo!'s autocomplete within the *onerror* attribute of an *img* HTML tag with a bad *src* attribute. We found that when the user moves the cursor over the autocomplete dialog, possibly with the keyboard when moving down toward other autocomplete suggestions, the malicious script is automatically executed. Such a cross-site scripting attack can be abused in many ways, e.g., to take-over the account. The attack exploits a combination of two vulnerabilities. The first is the lack of input sanitation, which allows cross-site scripting but requires the user to search for a script that will attack herself. The second vulnerability is the autocomplete-suggestions based on cross-site historical searches.

The combination of autocomplete injection attack with this innocuous XSS, results in a severe XSS vulnerability that can be exploited easily by an attacker.

7 Defenses

Web-services that offer autocomplete mechanisms should defend against the autocomplete injection attack. Autocomplete injection attacks are based on sending cross-site search requests, i.e., search requests initiated by a third-party rogue website; these requests are regarded by the web-service as searches performed by the current user (identified by cookie). A simple solution is to use CSRF defenses (see below), and prevent cross-site search requests. If cross-site search requests are considered useful and should be permitted, the websites should not take these searches into account as part of the 'history' of searches by the user. In particular, they should not let these searches influence the history-based autocomplete suggestions.

Web-services can detect most cross-site requests by inspecting the Referer or Origin HTTP request headers. However, these headers are not always sent, often due to filtering by client-tools for privacy; this could foil reliance on these headers to detect cross-site requests. As a solution, web-services can simply ignore searches that do not contain Referer or Origin headers, when determining the history-based autocomplete suggestions.

Alternatively, web-services can use other cross-site request forgery (CSRF) countermeasures. (See Overview [28]). The most popular active defense taken by a website is anti-CSRF tokens. In this basic defense against CSRF, an unpredictable token is sent with a request from the web-service, and is then validated by the server. Because the attacker cannot forge a token, she cannot send a request from another site that will pass the validation on the web-service side. Other CSRF countermeasures are discussed in [5,18,30].

Several of the most effective autocomplete injection attacks make use of *homographs*. Web-services can easily detect potential homographic attacks.

We believe that there will be no noticeable overhead in performance or loss-of-usability, due to blocking such homographic search strings (at least from the log of search strings).

Unfortunately, our experience shows that some web-service operators are reluctant to fix autocomplete-injection vulnerabilities. This motivates the development and use of client-side defenses. In particular, we recommend that, by default, browsers block cross-site requests that are suspected of being homographic attacks. Similar to web-services, we believe browsers can also detect such attacks with no noticeable 'costs', in terms of performance or usability. More advanced client-side defenses against CSRF attacks may also be applicable, e.g., [5].

8 Conclusions

Popular web-services offer autocomplete suggestions based on the history of user search strings; this saves users the effort of repeating a previous search. Additionally, web-services often allow cross-site queries, i.e., queries initiated by third-party sites, as long as the queries do not result in 'change of state'. More specifically, cross-site search-requests are allowed by many web-services, since they are considered 'harmless'.

Our experiments show that several attacks are feasible by exploiting the combination of the autocomplete mechanism, based on previous search strings, and the use of cross-site search requests. Specifically, we demonstrate how this facilitates autocomplete-based phishing, framing, illegitimate content-promoting and, at least in Yahoo!, persistent cross-site scripting. We propose both browser-based and web-server-based defenses against the autocomplete injection attack. We hope this paper will call attention to the problem and help address this potential vulnerability. It is also our goal to raise awareness regarding the general risk of permitting 'seemingly harmless' cross-site requests.

Acknowledgments. This work was supported by grant 1354/11 from the Israeli Science Foundation (ISF), and by grants from the Israeli Ministry of Science, Technology and Space.

References

1. Helou, A.J., Scott, T.: Multilingual web sites: Internationalized Domain Name homograph attacks. In: 12th IEEE International Symposium on Web Systems Evolution (WSE), pp. 89–92. IEEE (2010)
2. Alexa.Top Sites in United States, April 2016. http://www.alexa.com/topsites/countries/US
3. Anson, D., Moist, P., Przywara, M., Wells, H., Saylor, H., Maxime, H.: The effects of word completion and word prediction on typing rates using on-screen keyboards. Assistive Technol. **18**(2), 146–154 (2006)
4. Bortz, A., Boneh, D.: Exposing private information by timing web applications. In : Proceedings of the 16th International Conference on World Wide Web, pp. 621–628. ACM (2007)

5. Czeskis, A., Moshchuk, A., Kohno, T., Wang, H.J.: Lightweight server support for browser-based CSRF protection. In: Proceedings of the 22nd International Conference on World Wide Web, pp. 273–284 (2013)
6. Dover, D., Dafforn, E.: Search Engine Optimization (SEO) Secrets. Wiley Publishing (2011)
7. Ferguson, A.J.: Fostering e-mail security awareness: the west point carronade. EDUCASE Quarterly (2005)
8. Gelernter, N., Grinstein, Y., Herzberg, A.: Cross-site framing attacks. In: Proceedings of the 31st Annual Computer Security Applications Conference, pp. 161–170. ACM (2015)
9. Gelernter, N., Herzberg, A.: Cross-site search attacks. In: Proceedings of the 22nd ACM Conference on Computer and Communications Security, CCS 2015, pp. 1394–1405 (2015)
10. Google. Google Search Autocomplete (2014). https://support.google.com/websearch/answer/106230?hl=en
11. Greenberg, M.S., Wilson, C.E., Ruback, R.B., Mills, M.K.: Social and emotional determinants of victim crime reporting. Soc. Psychol. Q. **42**, 364–372 (1979)
12. Herzberg, A., Jbara, A.: Security and identification indicators for browsers against spoofing, phishing attacks. ACM Trans. Internet Techn. **8**(4), 16:1–16:36 (2008)
13. Holgers, T., Watson, D.E., Gribble, S.D.: Cutting through the confusion: a measurement study of homograph attacks. In: USENIX Annual Technical Conference, General Track, pp. 261–266 (2006)
14. Irani, D., Balduzzi, M., Balzarotti, D., Kirda, E., Pu, C.: Reverse social engineering attacks in online social networks. In: Holz, T., Bos, H. (eds.) DIMVA 2011. LNCS, vol. 6739, pp. 55–74. Springer, Heidelberg (2011)
15. Jagatic, T.N., Johnson, N.A., Jakobsson, M., Menczer, F.: Social phishing. Commun. ACM **50**(10), 94–100 (2007)
16. Manico, J., Williams, J., Mattatall, N.: Cross site scripting prevention cheat sheet, March 2016. https://www.owasp.org/index.php/XSS_(Cross_Site_Scripting)_Prevention_Cheat_Sheet
17. Grossman, J.: I know what websites you are logged-in to (Login-Detection via CSRF) (2009). http://blog.whitehatsec.com/i-know-what-websites-you-are-logged-in-to-login-detection-via-csrf/
18. Jovanovic, N., Kirda, E., Kruegel, C.: Preventing cross site request forgery attacks. In: Securecomm and Workshops, pp. 1–10. IEEE (2006)
19. Kidd, R.F.: Crime reporting. Criminology **17**(3), 380–394 (1979)
20. Klein, A.: DOM Based Cross Site Scripting or XSS of the Third Kind. Technical report, July 2005. http://www.webappsec.org/projects/articles/071105.shtml
21. Lu, L., Perdisci, R., Lee, W.: Surf: detecting and measuring search poisoning. In: Proceedings of the 18th ACM Conference on Computer and Communications Security, pp. 467–476. ACM (2011)
22. Milletary, J., CERT Coordination Center: Technical Trends in Phishing Attacks (2005). Accessed 1 Dec 2007
23. Parmar, B.: Protecting against spear-phishing. Comput. Fraud Secur. **2012**(1), 8–11 (2012)
24. Ruderman, J.: Same origin policy for javascript (2001). https://developer.mozilla.org/En/Same_origin_policy_for_JavaScript
25. Rydstedt, G., Bursztein, E., Boneh, D., Jackson, C.: Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In: IEEE Oakland Web 2.0 Security and Privacy (W2SP), pp. 1–13 (2010)

26. Stamm, S., Sterne, B., Markham, G.: Reining in the web with content security policy. In: Rappa, M., Jones, P., Freire, J., Chakrabarti, S. (eds.) Proceedings of the International Conference on World Wide Web, pp. 921–930. ACM (2010)
27. Tam, C., Wells, D.: Evaluating the benefits of displaying word prediction lists on a personal digital assistant at the keyboard level. Assistive Technol. **21**(3), 105–114 (2009)
28. The Open Web Application Security Project. Cross-Site Request Forgery (2010). https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)
29. Xing, X., Meng, W. , Doozan, D., Snoeren, A.C., Feamster, N., Lee, W.: Take this personally: attacks on personalized services. In: Proceedings of the 22nd USENIX Conference on Security, pp. 671–686. USENIX Association (2013)
30. Zhou, M., Bisht, P., Venkatakrishnan, V.N.: Strengthening XSRF defenses for legacy web applications using whitebox analysis and transformation. In: Mathuria, A., Jha, S. (eds.) ICISS 2010. LNCS, vol. 6503, pp. 96–110. Springer, Heidelberg (2010)