Cross-Site Search Attacks

Nethanel Gelernter
Department of Computer Science
Bar Ilan University
nethanel.gelernter@gmail.com

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

ABSTRACT

Cross-site search (XS-search) attacks circumvent the same-origin policy and extract sensitive information, by using the *time* it takes for the browser to receive responses to search queries. This side-channel is usually considered impractical, due to the limited attack duration and high variability of delays. This may be true for naive XS-search attacks; however, we show that the use of better tools facilitates effective XS-search attacks, exposing information efficiently and precisely.

We present and evaluate three types of tools: (1) appropriate *statistical tests*, (2) *amplification* of the timing side-channel, by *'inflating' communication or computation*, and (3) optimized, tailored *divide-and-conquer* algorithms, to identify terms from large 'dictionaries'. These techniques may be applicable in other scenarios.

We implemented and evaluated the attacks against the popular Gmail and Bing services, in several environments and ethical experiments, taking careful, IRB-approved measures to avoid exposure of personal information.

Categories and Subject Descriptors

J.0 [**Computer Applications**]: General

Keywords

Side channel attacks; Web; Privacy; Security

1. INTRODUCTION

Users of cloud and web services depend on the provider for their privacy. The provider is trusted not to make unauthorized use of the user's data and to protect the data against unauthorized access by third parties. Extensive research efforts are directed at reducing or avoiding these potential privacy exposures, e.g., by encrypting the data kept by the provider, possibly using searchable encryption [5]. Many works even focus on hiding the access pattern, using private information retrieval (PIR) [8] or oblivious RAM (ORAM) [17] for read/write.

CCS'15, October 12–16, 2015, Denver, Colorado, USA.
© 2015 ACM. ISBN 978-1-4503-3832-5/15/10$15.00.
DOI: http://dx.doi.org/10.1145/2810103.2813688.

However, these approaches are irrelevant for search engines, webmail and other Software-as-a-Service sites, to which users knowingly disclose significant private information, with consideration of the provider's reputation and privacy policy, and the relevant regulatory framework. It is still critical to protect privacy against rogue third parties, mainly by using *https* to protect against eavesdropping and MitM attackers, and relying on the browser's Same Origin Policy (SOP) to prevent exposure to rogue third-party sites.

Previous research [7, 12] showed that eavesdropping and MitM attackers can still learn information, by observing the *amount* of communication. However, these impressive attacks have significant limitations: they require eavesdropping, and attacker must wait for the user to access websites providing specific, sensitive services (e.g., medical or financial).

We show that by using timing side-channel, information can be exposed by (weaker, common) *cross-site attackers*, which simply control a rogue website to which the user surfs, and do not have eavesdropping/MitM capabilities. Furthermore, XS-search attacks apply even to general-purpose services such as webmail and search-engines, and to queries initiated by the attacker - no need to wait for user to perform sensitive query.

XS-search attacks do not require circumvention of the SOP or exploiting (known) browser/site vulnerabilities. This is in contrast to the (very common) known rogue-site attacks, which mostly exploit browser and/or website vulnerabilities, e.g., *Cross-Site Scripting (XSS)*, *Cross-Site Request Forgery (CSRF)* attacks [11, 21, 29] and DNS-rebinding [20]. These attacks are well-known and hence usually fail against 'serious' sites and updated browsers.

In particular, sites usually prevent CSRF attacks, by denying cross-site requests that have a 'sensitive' state-changing impact. On the other hand, even important, security-savvy web-services allow requests from other sites, which do not involve (sensitive) state-changes, since such requests may have different legitimate purposes.

Security experts know that cross-site queries create a timing side-channel [4]. However, there are significant challenges in using such side-channel: the attack is limited to the (short) duration when the user 'visits' the rogue website, and must deal with unpredictable delays and imprecise measurements. Hence, many web security experts consider it impractical to abuse the timing side-channel, for effective extraction of sensitive information. In fact, after we disclosed our attacks to vendors, Google referred us to a blog-post [13] that discusses this threat. In spite of being aware of the theoretical risk, Google and other vendors did not consider the attack to be practical and to justify adoption of appropriate defenses. This may be compared to 'theoretical' attacks on cryptographic systems, which may indicate that the cryptosystem does not meet the theoretical definitions - but usually require significant additional research and development of clever techniques and complex tools, to turn into a practical attack. Indeed, there are significant challenges facing XS-search attacks, including:

Noise: Delays depend on several dynamically-changing factors such as congestion and concurrent processes in client and server.

Small sample size: the attacking web-site has to take advantage of the duration of the visit by the user, which is often just a few minutes. Furthermore, the attacker must use a limited rate of requests to avoid detection and blocking (e.g., by server's anti-DoS defenses).

Measurement: time measurements in scripts are often inaccurate. We saw differences of dozens of milliseconds between the times we measured (in script), and more precise measurements (e.g., in browser console). Some browsers provide scripts with a performance object that records the loading time; attacker may use it to further improve accuracy [26].

We show that in spite of these limitations, cross-site attackers are still able to efficiently and accurately extract a significant amount of sensitive information, even from major websites. Specifically, in this manuscript, we demonstrate the attacks against Gmail, the largest webmail service, and Bing, the second-largest search engine; see [27] for other vulnerable major sites, e.g., Facebook, Outlook and Yahoo!. For example, a visit of a minute suffices for the rogue site to find the user's first *and* last names, with 90% success rate; see Table 3. Other examples of sensitive information that may be exposed include:

Search history: find out terms searched by the victim (e.g., in Bing).

Relationships: identify people and organizations with whom the user corresponds, possibly in relation to specific terms (e.g., romantic or professional), dates, etc.

Detect sender/recipient, including bcc recipients: check if a specific identity was the sender/recipient of a particular email, when the message was sent/received, and identities of sender and (other) recipients (incl. bcc recipients).

Detect terms in (specific) messages and folders: check whether a particular term or phrase appears in a specified subset of Gmail messages, e.g., in a specific folder, sent to/from specific identities, containing some other term, or sent/received during a specified time interval. This can also be limited to frequently-appearing items.

Structured (sensitive) information: for example, credit card and phone numbers.

The exposed information can be abused in many ways, for example, to facilitate spam and phishing, including *automated spear-phishing*.

Cross-site search attack. XS-search attacks exploit differences in the loading time of responses without results, to these of responses with few or many results. The differences may be due to differences in the computation time, and/or to differences in the transmission and processing times, mainly depending on response length. Typically, to learn whether or not a *challenge* request yields results, the adversary sends the request with a *dummy* request, which is known to return an empty response. The assumption is that the measurements for both the requests will be similar, if and only if the challenge request also yields an empty response. See Figure 1.

For example, to test whether the name of a Gmail user is "Vic", the attacker will send two search requests: the challenge request will ask for emails in the Sent Mail folder that were sent from Vic, and the dummy request will similarly ask for messages sent from a random dummy string. While the dummy request is expected to always yield an empty response, the challenge request will return a different response if and only if the name of the victim is Vic.

To deal with the challenges facing practical XS-search attacks, as described above, we investigated and developed several types of tools:

Statistical tests: The right statistical test is critical for efficient, accurate exposure. We evaluated several tests, including 'classical' tests and 'tailored' tests which we developed, that provided significant improvements.

Response-inflate mechanisms, where the attacker significantly increases the *size* of one of the two response options. This may require the term to appear in multiple records within the private data. See Figure 1(a) and Section 3.

Compute-inflate mechanisms, where the attacker significantly increases the *computation load* of one the two response options. This usually works, even when the term appears only once within the private data. See Figure 1(b) and Section 4.

Divide and conquer XS-search algorithms: A single search request can be used to answer a Boolean question. However, it is desirable to solve more complex problems, like detecting the name of the victim out of a list of 2000 names. To further efficiently use these mechanisms to extract information from private records, we developed appropriate *divide and conquer algorithms* to identify relevant terms among many. With these algorithms, an attacker can efficiently search within a large set of terms by comparing, each time, between two (or a few) large subsets. See Section 5.

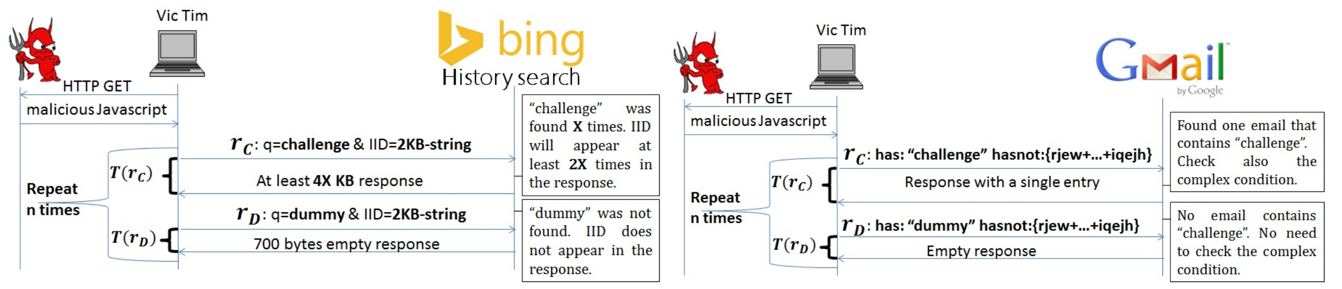
Contributions. We draw attention to the potential privacy exposure due to XS-search attacks. XS-search attacks are not trivial to launch; however, we present several types of tools that facilitate effective XS-search attacks: XS-search-optimized statistical tests, response-inflate and compute-inflate mechanisms, and divide-and-conquer XS-search algorithms (see brief descriptions above, and details within).

We evaluated these different tools extensively, using ethical, IRB-approved experiments, focusing on some of the most popular web-services.

The tools may be applicable to other scenarios. Specifically, our evaluation of different statistical tests, strengthens conclusions of [10], but further shows that in some practical scenarios, simpler tests can be used to efficiently extract information, and may allow the use of less samples.

1.1 Ethics and Disclosure

We implemented all of our attacks (in JavaScript), and validated them using experiments in different environments.


Vic Tim
HTTP GET
malicious Javascript
bing
History search
r_C: q=challenge & IID=2KB-string
At least 4X KB response
$T(r_C)$
Repeat n times
r_D: q=dummy & IID=2KB-string
700 bytes empty response
$T(r_D)$
"challenge" was found X times. IID will appear at least 2X times in the response.
"dummy" was not found. IID does not appear in the response.
Vic Tim
HTTP GET
malicious Javascript
Gmail by Google
r_C: has: "challenge" hasnot:{rjew+...+iqejh}
Response with a single entry
$T(r_C)$
Repeat n times
r_D: has: "dummy" hasnot:{rjew+...+iqejh}
Empty response
$T(r_D)$
Found one email that contains "challenge". Check also the complex condition.
No email contains "challenge". No need to check the complex condition.

(a) Response-inflate attack example (Section 3)

(b) Compute-inflate attack example (Section 4)

Figure 1: Examples of effective XS-search attack using the Response-inflate and Compute-inflate techniques. In the end of the attack, the adversary uses statistical tests to decide whether $T(r_C)$ and $T(r_D)$ were sampled from the same distribution.

All experiments were IRB-approved after careful design and evaluation, to avoid privacy exposures or other ethical issues. The experiments were carried out on 'real' (e.g., Gmail) user accounts and (e.g., Bing) browsing logs. We focused on the experiments that we could ethically perform on data from a significant number of (paid) volunteers, including students as well as users of the Amazon's Mechanical Turk service. In these experiments, we only 'expose' items that the users shared with us voluntary, such as their name or a random value they selected. This allowed us to validate correctness and to avoid unintentional privacy exposure. We also carried out several experiments on our own accounts to validate that the techniques expose 'real information', e.g., credit card numbers, passwords and names of contacts, locations, projects etc. We have alerted both Bing and Google, and they confirmed the attacks and adopted countermeasures. We also alerted additional popular websites which we found vulnerable to the attack.

1.2 Related Work

Timing and other side-channels were effectively used to circumvent many cryptographic and access-control defenses [3, 6, 10, 22, 23, 30]. Several works present cross-site attacks that expose the browsing-history of the user [14,31], or identify sites to which the user is connected concurrently [16]. Bortz et al. [4] showed that these techniques can also expose the number of items in shopping cart.

Cross-site side-channel attacks that extract information using database queries were presented by Futoransky et. al [15], assuming the ability to perform public queries over the database. However, their results were very limited with regards to the amount of information leaked (and that was in an 'ideal' setting of direct access to their own database). A related, well-known technique is *blind SQL injection*. However, this technique depends on SQL injection vulnerabilities and is not likely to apply to well-protected web services (e.g., see [9]). Furthermore, large web-services such as webmail and search engines rarely use SQL.

Evans discussed XS-search attacks in a blog-post [13], however, without evaluating its effectiveness or developing any tools to make the attack practical; indeed, from the responses of vendors, this attack was not considered practical.

2. XS-Search ATTACK

In this section we briefly introduce the XS-search attack and its use of timing side-channel. We first describe the XS-search attack, and then discuss techniques for analyzing the responses' loading times. In Sections 3 and 4 we discuss two 'inflation' techniques that improve the attacks by decreasing the number of requests required for the attacker to learn information.

2.1 Adversary Model & Attack Process

Despite the large amount of information that can be obtained through it, the XS-search attack can be launched by a weak adversary. This weak adversary runs a malicious website, but does not require eavesdropping or MitM abilities. From her website, the adversary can send HTTP requests and measure the time until a response is received. The measurement is done in the victim's browser, for example, using JavaScript. The attacker might attack casual visitors to her website, or use social engineering techniques to lure specific targets to visit the site.

The attacker's goal is to detect whether a search request has results or not. We denote a search request for which we want to learn whether it has results as the *challenge request* (r_C). The main idea of XS-search is as follows: the time it takes until an HTTP response is loaded by the browser actually leaks information about the response's content. In particular, the loading time of a response with search results (*full* response) and a response without them (*empty response*) is often different. The difference might be due to the processing time of the request or due to the size of the response. Hence, we concentrate on measuring the time required to receive a response to search requests.

Attack process. We assume that the attacker can send search requests that are replied with an empty response. This is a reasonable assumption because in most of the services, search requests for meaningless terms that a user is unlikely to ever use, are usually replied with empty responses. We call such a request a *dummy request* (r_D).

To launch the attack, the attacker sends multiple pairs of challenge requests (r_C) and dummy requests (r_D). To avoid cached response, the attacker concatenates a random dummy parameter to each request. We denote the loading time of responses for request r by $T(r)$. Based on the measured response time values, the attacker has to decide whether the challenge request r_C also resulted in an empty response (like r_D), or if it resulted in a non-empty response, i.e., there were some matching records.

The assumption behind the analysis of the results is that the values in $T(r_C)$ will be relatively similar to the values in $T(r_D)$, if r_C also receives replies with empty responses. On the other hand, we expect to see a greater difference between the distributions, if r_C is replied to with a full response, unlike r_D.

2.2 Analyzing the Measured Times

Given $T(r_C)$ and $T(r_D)$, the attacker's challenge is to decide whether r_C was replied to with a different response than r_D. This problem of deciding whether two sets of values were sampled from the same distribution is well known. We first describe classical tests for solving this problem and then additional simple tests we used.

2.2.1 *Classical Hypothesis Tests*

We build on the seminal work of Crosby, Wallach, and Riedi [10], who evaluated the use of timing channels as a means to perform cryptanalysis on a remote system. Crosby et al. evaluated several well-known hypothesis testing methods, including the classical and modified Student's t-test and the Mann-Whitney U test, and showed they were all significantly inferior to a new and simpler test they developed, the *box test*.

The box test receives as parameters a small interval of two percentiles $[a, b]$, where $0 < a < b$ and typically $b < 6\%$, and a pair of sets of measurements $\{(m_D^i, m_C^i)\}$, where m_D^i are measurements of timing for dummy request r_D, and m_C^i are measurements of timing for the challenge request being evaluated, r_C. The box test estimates that the m_D^i measurements are from the same distributions if there is overlap between the $[a, b]$ percentile values of the two sets $\{(m_D^i\}$ and $\{m_C^i)\}$.

We compared the results for the same set of well-known tests, together with the box test (slightly modified, see next paragraph), when applied to our scenario involving timing side-channel using remote measurements. However, in our case, the main impact was for non-cryptographic processing and communication delays, while for Crosby et al. [10] these are just noise to the signal (the cryptographic computation time). Our results show that, as in [10], the box test outperformed all the well-known statistical tests we compared against.

Since our results were good enough to allow the use of tiny samples, we had to slightly modify the box test. Namely, instead of checking for overlap between the $[a, b]$ percentile values of sets $\{(m_D^i\}, \{m_C^i)\}$, for some small $b < 6\%$, we checked for overlapping between the intervals $[m_D^i, m_D^j]$ and $[m_C^i, m_C^j]$, for low values $i < j$ (e.g., $0 \leq i < j \leq 2$). We denote the box test between the i-th and j-th lower values by BX_i^j.

In addition to the box test, we used the Apache Commons Mathematics Library [1] to perform the following 'classical' hypothesis tests: Student's t-test, Mann-Whitney U test (denoted MW), Wilcoxon signed-rank test, Kolmogorov-Smirnov (denoted KS) two sample test, and one-way ANOVA test [24]. Throughout the paper, we evaluate the timing results according to the box test, the KS test, and the MW test, which achieved better results than the other 'classical' hypothesis tests.

2.2.2 *Tiny-Sample Tests: MIN and AVG*

As Crosby et al. pointed out [10], basic statistical measures such as the median or the average, do not provide very good results. However, since we found that even tiny samples seem to have meaningful differences in our measurements, we defined and evaluated two very simple tests. In our experiments, these tests consistently *outperformed* the box test, as well as all classical statistical tests we compared against. Details follow.

Our new tests are based on Crosby et al.'s observation that the lower values in the samples give a better indication about the differences between the distributions. We define the $\mathrm{AVG}_{\text{-}p}^t$, which contains three steps: (1) From each sample, $T(r_C)$ and $T(r_D)$, remove the $p\%$ highest values. (2) Calculate the average of the remaining samples, $\mathrm{AVG}_{\text{-}p}(T(r_C))$ and $\mathrm{AVG}_{\text{-}p}(T(r_D))$. (3) Return true if and only if $\mathrm{AVG}_{\text{-}p}(T(r_C))/\mathrm{AVG}_{\text{-}p}(T(r_D)) > t$.

Similarly, we define the MIN_i^t test, where we compare to t the ratio between the lowest i-th values.

The main advantage of these tests is the fact that they are applicable and effective even for minimal samples. Another advantage is their easy evaluation, which allows the attacker to effectively run them at the client side using JavaScript. Since all these methods use a threshold, this poses a challenge of finding a good (or optimal) threshold. In spite of this, we found it easy to find thresholds that achieve better results than the box test and the 'classical' statistical tests. See for example results in Section 3 or Table 2 in Section 4.2. In practice, an attacker can find the threshold value by simulating the attack in a controlled environment.

3. Response-Inflate XS-Search Attack

The loading time of responses to search requests depends on the number of search results. It takes longer to prepare and transmit *long, full* responses that contain many search results, than *empty* responses indicating no results. A longer response means greater processing time and greater transmission time. Even if the response is sent compressed, there is still a difference in the size, and it takes longer to zip and unzip long responses.

Following the approach outlined in Section 2.1, the difference in time to receive a response may allow a cross-site attacker to detect whether the response was empty or not. Namely, the attacker sends the challenge request together with the dummy request, which is expected to yield an empty response, and compares the loading time of the two responses.

In this section, we present the *Response-inflate XS-search attack*, a technique that makes it easier to distinguish between the loading time of empty and full responses, allowing the attacker to find the correct answers to Boolean queries on the sensitive data of the user in the records of the web-service. In Section 5, we use such techniques as building blocks, to answer more complex queries, e.g., to efficiently find out the user's name, phone-number, and other information.

The idea of the attack is to increase the difference between the size of empty and full responses. We begin by presenting some methods of response inflating, which we found applicable to many web-services (see [27]), and then we give specific details for two example sensitive data services: the Bing history logs and the Gmail email archive.

3.1 Response-Inflate XS-Search Attack: Methods

In many web-services, each request may contain a parameter that is copied, at least once - often more - for each entry in the response. Furthermore, a response may often contain many entries, typically, one entry per every record which fits the criteria in the request. By sending a relatively-long parameter, the length of the response is significantly influenced by the number of entries, which often allows the attacker to distinguish between empty responses, i.e., no-match, vs. responses with significant number of matches. This is facilitated by the fact that many web-services do not explicitly limit the length of such parameters, and in particular, many allow parameters of over $2KB$.

The number of times that such parameters are copied, may depend on properties of the entry in the response, which may allow an attacker to detect the relevant property. For example, when the Bing service receives a *more* request to Bing, it copies the value of the *IID* parameters *three times* for each query result which the user 'followed' (clicked on), and *twice* for each 'orphan' response (where the user did not follow on any result); details in subsection 3.2.

3.2 Response-Inflate XS-Search Attack on Bing's Search History

Bing's search history is an example of a service vulnerable to the Response-Inflate XS-Search Attack. Many web-services maintain a *log* containing 'history-records' listing the operations done by each user. Logs often contain sensitive private information and should not be made publicly available.Services often allow users to view and search within their search log. In particular, Bing (and others such as Google) permits cross-site search requests from a third-party site, i.e., they do not apply CSRF-restrictions for such requests. In both Bing and Google, the ability to do cross-site history searches may allow an attacker to learn which search terms were used in previous searches by the user; we focus on Bing.[1]

Terminology: requests vs. queries. To avoid confusion, we use the term *request* for cross-site requests for search within the user's search history as kept by Bing, and the term *query* for search queries done by the user (entries in the search history). Namely, the attacker sends search *requests* that return search *queries*, sent in the past by the victim.

Bing History Feature. Bing supports two types of requests for searching through its search-history: *search* requests and *more* requests. The Bing client-side script sends 'search' requests based on the search parameters entered by the user, and sends 'more' requests when the user scrolls-down the page to see additional results.

Bing responds to both search and more requests with a list containing up to twenty entries from the user's search-history. For *fruitful search queries*, i.e., queries for which the user clicked on some of their results, Bing uses one entry for each clicked-result. For *orphan search queries*, i.e., queries for which the user did not click on any result, Bing uses only a single entry.

The twenty queries returned for search requests are the most recent search-queries of the user (that fit the search terms given by the user). To allow Bing to provide the 'next recent' entries, the 'more' requests have a parameter t, which specifies a specific time; Bing replies with the most recent history entries that are dated *prior* to time t. Bing allows this parameter to be set to a future (or current) time, and in this case simply returns the latest results (similar to the 'search' query).

[1]The attack is not as relevant in Google since Google requires users to re-authenticate in order to search their history records; we believe that in practice, this would usually foil the attack.

The maximal length of both the *search* and *more* requests is $2KB$. Each of these two requests contains a parameter that is copied, at least once, for each entry in the response; its length can be almost as long as the $2KB$ length limit. In the response for a *search* request, this parameter is the *FORM* parameter. The value given in this parameter is copied once for each query appearing in the response, plus four more times. In the response for a *more* request, the parameter is the *IID* parameter. The value given in this parameter is copied *twice* for each entry representing an *orphan* query, and *three times* for each of the multiple entries of a *fruitful* query.

Finally, there is a significant difference in the size of empty responses for *search* versus *more* requests. For *search* requests, the size of empty responses is about $50KB$, while for *more* requests, the size of such responses is less than 1KB. The smaller size of empty responses and the higher inflation ratio, make the *more* request more suitable for the Response-inflate XS-search attack.

Attack details. Consider a request that is replied to with a full response containing X *orphan* queries and Y entries for all the *fruitful* queries together. By using *FORM* and *IID* parameters of 2000 bytes each, the size of the response for the *search* request is inflated by nearly $2000 \cdot (X + Y)$ bytes compared to the empty response. Moreover, the size of the response for the *more* request is inflated by nearly $2000 \cdot (2X + 3Y)$ bytes. This inflation is exacerbated since Bing does not compress (zip) their responses.

3.2.1 *Bing-History Response-Inflate XS-Search Attack: Environments*

To analyze the effectiveness of the Bing-history attack, we applied it on a Bing account we have set-up, in three different environments:

1. *High speed wired connection (HS).* We used a server in our lab with measured 95 Mb/s download speed.
2. *Home network wireless connection (HN).* We used a laptop connected to a home wireless network. The measured download speed was $12 - 15$ Mb/s.
3. *Open wireless network (OW).* We used a laptop connected to our university's wireless network. During the tests, we measured a download speed that was generally around 3.5 Mb/s (the highest measured rate was 4.05, and the lowest 1.59).

Note that the HS and the OW environments present some challenges for the attack. In high-quality high-speed connections, although the variance of the loading time is expected to be low, the difference in the transmission time of two objects of different sizes is also lower. In the OW environment, although the download speed is low, which makes the difference in the loading times greater, the variance in the loading times is higher due to the frequent changes in the load from other computers.

We tested the detection of term T_i that appears i times in the Bing history, for $i \in \{1, 3, 5, 10, 20, 50\}$. For each environment and each T_i, we took samples of at least 500 pairs of r_C and r_D, such that r_C asks for T_i and r_D is a dummy request (which is certain not to exist in the Bing search log). Then we separated the measured samples into groups of $n \in \{5, 10, 15, 20, 25\}$ consecutive pairs, and used

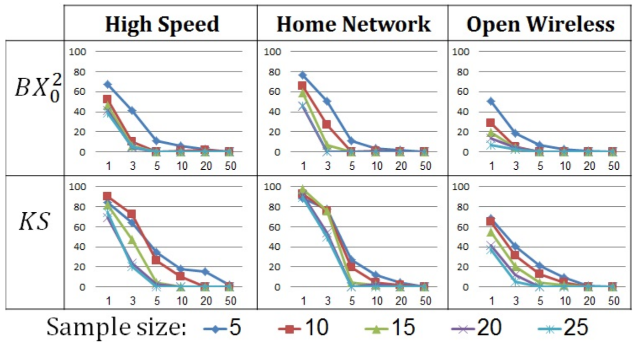


Figure 2: False negative rate (%) for BX_0^2 and KS tests in different environments, as a function of the number of times the term (T_i) appears in history ($i \in \{1, 3, 5, 10, 20, 50\}$).

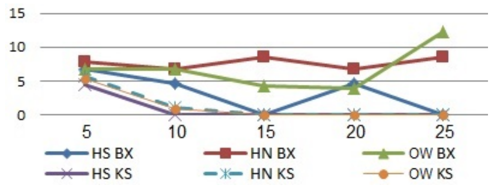


Figure 3: False positive rate (%) for the BX_0^2 and KS tests in different environments, as a function of the sample size.

several tests to decide whether r_C was replied with results or not. Based on the results, we measured the false-negative (FN) rate of the methods. Similarly, for each environment, we sent pairs of r_D and r_C requests, such that r_C is another dummy request, to evaluate the false-positive (FP) rate. Figure 1(a) depicts the attack.

3.2.2 *Results: Classical Hypothesis Tests*

For each environment, T_i and n, we tested all the tests mentioned in Section 2.2.1. The box test [10] and the Kolmogorov-Smirnov (KS) test with $\alpha = 0.01$ achieved the best results. In the box test, we checked for overlapping in the range between the three shortest times measured in each sample; namely, we used BX_0^2. We checked several ranges, and this range gave the best results for most of the cases.

Figure 2 shows the FN rate of KS and BX_0^2 in each of the environments, for several sample sizes, as a function of the frequency of the searched term in the history. The graphs show that the FN rate decreases for terms that appear more times in the history. The box test achieves better results than KS and the other tests. Specifically, it achieves very good results for terms that appears at least 5 times in the history, even when using small samples with 5 pairs of requests.

Figure 3 shows the FP rate of each of the tests in the different environments, as a function of the sample size.

3.2.3 *Results: MIN and AVG Tiny-Sample Tests*

We next evaluate the new tests which we introduced in Section 2.2.2; we show that with the right parameters, these simple tests can achieve excellent results. In Figures 4 and 5 we present the FN and FP rate of the $MIN_0^{1.05}$ and $AVG_{-25}^{1.05}$ tests. The threshold (1.05) was chosen based on a test that simulated the attack in the HN environment. Although this threshold is not optimal, we can see that in the HN and OW

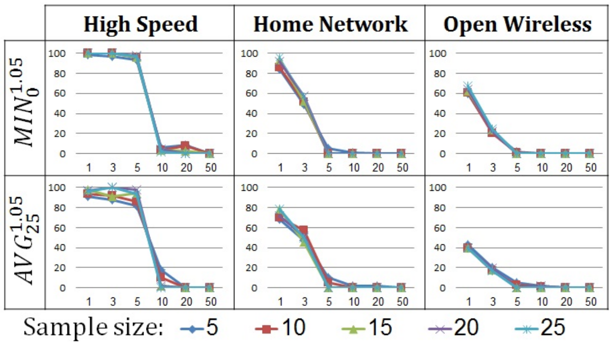


Figure 4: False negative rate (%) for the $AVG_{-25}^{1.05}$ and $MIN_0^{1.05}$ tests in different environments, as a function of the number of times the term (T_i) appears in history ($i \in \{1, 3, 5, 10, 20, 50\}$).

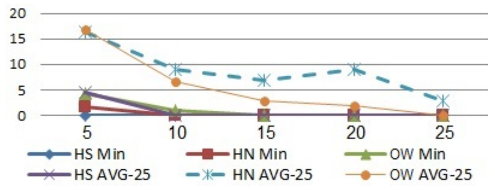


Figure 5: False positive rate (%) for the $AVG_{-25}^{1.05}$ and $MIN_0^{1.05}$ tests in different environments, as a function of the sample size.

environments both the tests achieve very good results in detecting terms that appear at least 5 times in the history with samples of 5 requests. In particular, $MIN_0^{1.05}$ achieves lower FN and FP rates than BX_0^2. To achieve better results for the HS environment, we should use a lower threshold (e.g., 1.02). In practice, the adversary can estimate the connection quality and speed by downloading a large object, and choose the threshold accordingly. However, this procedure increases the time required for the attack. Additional details and extended analysis appear in [27].

3.2.4 *Response inflation effect*

To test the effect of the response inflation, we repeated our evaluation experiment in the OW environment with simple (not inflated) requests. The results showed that although it is possible to distinguish between empty and full responses, a larger sample size is required. We also observed that the response inflation decreases the FN when detecting a full response that contains fewer entries. Figure 6 depicts the FN difference according to the box test (BX_0^2) for simple and inflating requests. We extend the comparison in [27].

3.3 Response-Inflate XS-Search Attack on Gmail

Google's Gmail is the most popular email service today. We observed that it is possible to extract information from Gmail accounts by measuring the times of cross-site search requests and further inflating the difference in the sizes of full and empty responses. Compared to the attack on Bing, the inflating process is weaker due to compression of the responses. Because the response inflation is done similarly to the attack on Bing, we briefly describe the response inflation and further elaborate in [27].

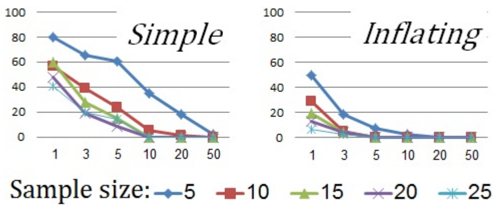


Figure 6: False negative rate (%) for the BX_0^2 using simple and inflating requests, for different sample sizes as a function of the number of times the term (T_i) appears in history ($i \in \{1, 3, 5, 10, 20, 50\}$).

Type (view mode)	No results	20 entries	50 entries
Simple (standard)	147 (40.6)	166 (44.5)	166 (44.5)
Simple (HTML)	10 (3.4)	20.2 (5.7)	35.3 (8.5)
Inflated (HTML)	106 (11)	303 (21.1)	575 (25)

Table 1: Comparison of response size for Gmail search query in the different modes. The size after compression appears in parentheses. The values are in KB.

Information leakage. The difference in the processing time of full and empty responses allows an attacker to answer Boolean questions, represented as search queries, about the data that often appears in Gmail accounts. Among the things that usually appear many times in Gmail accounts are the name of the user, frequent contacts, data from the email signature, and companies and services that send periodic updates or reports. Boolean questions about whether the name of the user is X, whether she has a contact named Y, or is a client of bank Z, can be sent as a search query for emails sent from X, sent or received from Y, or received from the known 'no-reply' email address of bank Z.

Response inflation. Similarly to what we did for Bing, we inflated the search query to its maximal size, which, in Gmail, is 8KB. In Gmail's basic HTML view [19], the search query appears once for each entry in the response; this dramatically inflates the HTTP body of a full response over an empty response. The default maximal number of entries in a response is 50. Although Gmail sends the responses zipped, and hence the improvement is not as significant as in the Bing history attack (see Section 3.2), it takes longer to create, zip, and unzip longer responses. A comparison between the simple and the inflating requests for launching effective timing attacks is discussed in [27].

4. Compute-Inflate XS-Search Attack

The Response-inflate XS-search attack is based on causing a significant difference in the *length of the response* due to its dependency on the number of terms returned. For this difference to be significant, the number of terms should be substantial. In contrast, in this section, we present the Compute-inflate XS-search attack, which can effectively check in the private data for the existence of records that meet specific search criteria - *even a single record*. The Compute-inflate XS-search attack is based on causing a significant difference in the *time* required to process the search queries; this difference can often be significant for carefully engineered queries, even when the queries return roughly the same *amount* of information. This exploits the fact that the time to process queries may vary significantly, depending on specific conditions.

Specifically, the Compute-inflate XS-search attack is applicable for the many web services satisfying the following conditions:

1. **Cross-site queries allowed:** service can be invoked directly from a different-origin webpage, i.e., does not restrict service to calls issued by same-origin pages, using the *referer* header or other CSRF-prevention mechanisms.
2. **Conjunction allowed:** queries may be constructed as a conjunction of two (or more) terms.
3. **Known hard-to-compute terms:** it is easy to find hard-to-compute (computationally intensive) terms.
4. **Early abort:** When the search query is a conjunction of easy-to-compute and hard-to-compute terms, the easy term(s) are computed first. Alternatively, processing is done on all terms in parallel, or sequentially but in a known order among terms (e.g., from left to right). Furthermore, once any term(s) return *false*, then processing is aborted (returning *false*) without waiting to complete the computation of the hard-to-compute terms.

Consider a search query containing the conjunction $\sigma \wedge \theta$, where σ is an easily-computable search term and θ is a hard-to-compute term. From the 'early abort' condition, if there is no record matching σ, then the service will immediately return a negative response (no matching record). Conversely, if there is a matching record, i.e., σ resolves to *true*, then the service would proceed to evaluate the 'hard' term θ, and return results only after that evaluation is completed. Comparing the computation time for $\sigma \wedge \theta$ versus the computation time for $\bar{\sigma} \wedge \theta$, provides a timing side-channel that allows an attacker to determine whether the private data-set contains records matching the σ term. Depending on the web-service and the search terms it supports, this may allow very precise and privacy-invasive queries.

We demonstrate the effectiveness of the attack for such precise resolution of queries over the Gmail email-archive of the user. Gmail allows third-party (cross-site) queries and supports complex combinations of a rich set of query terms. Gmail uses different optimizations to efficiently evaluate most typical search queries. This includes searches for specific strings within the message body (using inverted index) and conditions computed over meta-data fields such as sender, recipients (including 'bcc' recipients for messages sent by the user), date, and subject. On the other hand, Gmail allows queries that are rather complex and may result in significant processing, such as the disjunction of multiple strings or conjunctions of 'negative' (*exclude*) terms, and the search for exact strings.

Gmail, like most services, does not disclose their search algorithms. However, as we demonstrate, attackers can experimentally validate that the above requirements hold, and in particular find appropriate hard-to-compute terms. For Gmail, we found that one easy way to construct a 'hard-to-compute' term is using a conjunction of many *exclude* terms.

4.1 Cross-site Existence Queries on Gmail

To be vulnerable to Compute-inflate XS-search attack, a service must meet the four conditions described above. Gmail satisfies the first two conditions: it allows cross-site queries and supports the conjunction of queries. We now discuss the other two requirements: known hard-to-compute terms and early abort. Given an easy-to-compute query σ, where the attacker wants to find whether it matches at least a single email, the attacker needs to send the query $\sigma \wedge \theta$, such that θ is hard-to-compute. An easy-to-compute query is simply a query that asks for a single term. For example, a query that asks for emails that contain some term or that were sent to/from some person.

Hard-to-compute query. As the hard-to-compute query θ, we used the *'has not'* operator [18] for searching emails without some list of terms. Specifically, we used a θ that asks for emails which do not contain a long list of random strings that probably do not appear in a typical email. Following Gmail's advanced search restrictions, we used a list of almost 1000 four-character random strings composed of characters and letters.

Early-abort and hard-to-compute evaluation. Before we evaluated the use of many-*has not* query for the Compute-inflate XS-search attack, we verified two basic claims: (1) When appending the *has not* operator with many values to a query, it is easier to distinguish between a response that contains one message and a response that contains no messages. (2) The use of the *has not* operator with many values, makes the distinction easier.

To test the claims, we sent an email from a Gmail account with very low activity and only few sent emails; in such an account it is faster to find emails. We launched the attack described in Section 2.1 with three different types of queries: (1) complex search query as described above, (2) simple search query, and (3) complex search query with one long *has not* term composed of the concatenation of all the *has not* terms in the first complex query.

We took measurements of at least 500 pairs of requests (r_C and r_D) of each type in the HN environment. We first used an r_C that was replied to with a single email to measure the FN rate; we then repeated the process with an r_C as a dummy query to measure the FP rate. While using the simple requests, it was hard to detect a single message in the response; this was the case even when using a sample of $n = 50$ requests pairs (almost 50% FN). With complex queries, several tests achieved *zero false positives and zero false negatives*, even with $n = 25$. This substantiated the first claim. The results using the one-long-*has not* queries were better than the simple search queries, yet far from the zero FN and FP of the complex query. This confirmed our second claim, as the responses for both the queries were of the same length. Repeating the comparison between simple queries and complex queries on real users gave the same indication; see below.

4.2 Compute-Inflate Attack on Gmail: Evaluation

To evaluate the Compute-Inflate Attack, we conducted several experiments for participants using their active Gmail account. These were designed with IRB approval. We describe here only the first experiment, which tested the effectiveness of answering a Boolean question about whether there is some email that contains a specific sentence. We also repeated the evaluation we did in Section 4.1 to show that queries of the form $\sigma \wedge \theta$, where θ is a hard-to-compute query (see Section 4.1), are much more effective for learning about the existence of content than simple queries (σ). We conducted a similar experiment for detecting a single email by its recipient or subject, with different users, and received similar results; see [27].

Our content-detection experiment had two goals: (1) show that using Compute-inflate XS-search attack it is possible to learn about the existence of a sentence in Gmail account folders, even if it appears only once, and (2) confirm that the conjunction with a hard-to-compute query makes the distinction easier. We conducted the experiment with the participation of 138 Gmail users; 90 of them were from our university and 48 were Amazon Mechanical Turk workers. Each participant ran the experiment from her own computer, using the Internet connection available to her.

Test	Complex queries				Simple queries			
	$n = 25$		$n = 50$		$n = 25$		$n = 50$	
	FN	FP	FN	FP	FN	FP	FN	FP
$\text{Avg}^{1.02}_{-25}$	6.2	10.5	2.9	6.5	53.6	9.8	54.3	8.7
$\text{Min}^{1.02}_{2}$	9.1	15.2	5.1	8.7	59.4	9.1	60.9	7.2
$\text{Min}^{1.02}_{4}$	9.1	11.2	7.2	8	60.5	8.3	65.9	8
$\text{Min}^{1.02}_{6}$	8	11.6	7.2	8.7	56.5	5.8	62.3	5.8
MW	17	7.6	9.4	5.1	50	10.1	39.1	13
KS	17	8	8.7	8	47.8	10.1	35.5	14.5
BX^{2}_{0}	16.7	8.7	12.3	7.2	42	11.6	36.2	12.3
BX^{3}_{1}	11.2	15.9	7.2	12.3	32.6	15.9	25.4	18.1

Table 2: Analyzing false negative (FN) and false positive (FP) for Compute-inflate XS-search attack, to detect a sentence that appears only once in Gmail mailbox; complex vs. simple queries. See test descriptions in Section 2.2.

During the experiment, we asked the participants to send an email with a single *challenge-sentence*: "I like *random-animal*" concatenated to some random number. For each participant, we chose a *random animal* among 32 options. The concatenated random number was used to make absolutely sure the sentence appears only once in the tested Gmail account.

To measure the false negative (FN) rate of the attack, we sent a sequence of 50 pairs of requests; the first request asked for emails in the Sent Mail folder that contain the challenge-sentence, and the second asked for a similar sentence that did not exist. To measure false positive (FP) rate, we repeated the process, but this time, the first request in the pair asked for another non-existent sentence.

We measured the FN and FP for the attack using simple Gmail queries and complex queries. The complex queries were created by appending to the simple query a *has not* operator with a list of many terms, as discussed above (see also Figure 1(b)). We used the tests mentioned in Section 2.2 to analyze the times and to conclude whether the participant had an email containing the challenge-sentence.

We analyzed the times for samples of $n = 50$ pairs and for $n = 25$ pairs. Table 2 presents the analysis of the measurements according to several tests; the results of all the tests we measured appear in [27].

The analysis shows that complex queries significantly improve the effectiveness of the attack compared to simple queries. The results show that using 25 pairs of search queries, it is possible to identify a single email by its content with reasonable false positive (FP) and false negative (FN) rates. By increasing the sample size to 50, we further improved the results. The best results were achieved by the $\text{AVG}^{1.02}_{\text{-}25}$ test, described in Section 2.2.2.

5. EFFICIENT TERM-IDENTIFICATION

In a *term-identification query*, the attacker has a large set S containing n potential search terms (e.g., person/location/project/other names, phone-numbers, credit-card numbers, passwords), and wants to identify *which*, if any, of the terms appear in the private data records (many times, in specific fields, or even just in one arbitrary record). In this section, we present three algorithms that allow a cross-site attacker to efficiently perform term-identification queries against the private data records of the user, kept by a web-service. The algorithms build upon the *single-term query* algorithms of Sections 3 and 4, where the attacker only learns if the private data contains records matching one given search term. A naive term-identification method, is to perform n single-term queries. However, n is often large (e.g., the number of possible names), i.e., this attack may require the user to remain connected to the rogue site for unreasonably-long period, and therefore this naive method may have limited value in practice.

In contrast, we present three *divide and conquer* algorithms to efficiently run term-identification queries, when given large set of potential terms. In order to apply 'divide and conquer', we assume that the attacked web-service allows search terms that are disjunctions (logical *or*) of other terms. Namely, if τ and τ' are search terms, then $\tau \vee \tau'$ is also a search term. In particular, the Gmail webmail service, which we used to evaluate our attacks, supports such composition of terms (using the *OR* operator), as well as a rich set of other search operators [18].

We consider two variants of the term-identification queries: *multiple-term identification queries*, whose goal is to identify all or most search terms that have matching records; and *identify any query*, whose goal is to identify just one of the search terms that have matching records. For example, the *multiple-term identification* query is appropriate when searching for the set of correspondents or keywords used in emails sent/received by the user. The *identify any* query is appropriate when searching for a specific value (e.g., to find out the address the user includes in her 'signature' appended to outgoing emails).

In Subsections 5.1 and 5.2, respectively, we present the MTI and OMTI algorithms, both for the *multiple-term identification queries* variant. In Subsection 5.3, we present the ATI algorithm for the *identify any query* variant. In Subsection 5.5, we experimentally evaluate the algorithms; Note that when the number of terms is small (e.g., two), the any-term algorithm can also be used by running it twice. (In the second run, the item identified in first run is excluded.)

However, all three algorithms may fail, when the attacker searches for terms from a really huge set such as phone-numbers, credit-card numbers or passwords. In subsection 5.4 we show that often, even in such cases, we can find the relevant terms, by taking advantage of common *properties and formats* of the relevant terms.

5.1 The Multiple-Term Identification (MTI) Algorithm

The MTI algorithm (Alg 1) is a simple, generic divide-and-conquer algorithm, whose goal is to identify *all or most* of the terms, within a given set of terms S, that have some property θ. In this paper, the relevant property is the existence of records matching θ in the private user's data searched by the web service.

To find the terms $s \in S$ that have the property θ, i.e., $\theta(s) = \mathit{True}$, the MTI algorithm is given access to a *noisy, probabilistic* test T_θ. Given a set of terms $S' \subseteq S$, the test T returns *True*, with a high probability, if and only if some $s \in S'$ satisfies θ. MTI uses T_θ in a modular 'black-box' manner; it receives T_θ as a parameter. MTI also receives parameter μ, specifying the maximal number of terms allowed in set S', which MTI gives to T_θ. For example, in Gmail, search queries are limited to about 8KB.

Specifically, in the XS-search attacks, the test T_θ sends challenge requests asking for records that match any of the terms in S', and dummy requests that similarly ask for records that match $|S'|$ dummy values, as described in Section 2.1. The test then analyzes the difference between the loading times of the responses for both the request types and decides whether their responses are different.

Algorithm 1 Given a set of terms S and test T, algorithm MTI outputs terms in S that appear in the search results. Parameter μ is the maximal number of terms per query.

```
MTI(S, T, μ):
    Return ← $\bigcup_{i=1}^{\lceil |S|/\mu \rceil} MTIr\left(\left(\cup_{j=\mu(i-1)}^{\mu \cdot i-1} S[j]\right), T\right)$

MTIr(X, T):
  if $T(X)$ then
    if $|X| = 1$ then return $X$.
    else
      $X_L \leftarrow X\,[1, \ldots, \lceil |X|/2 \rceil]$
      $X_R \leftarrow X\,[\lceil |X|/2 \rceil + 1, \ldots, |X|]$
      return $MTIr(X_L, T) \cup MTIr(X_R, T)$
  else
    return $\emptyset$
```

Comment: T_θ may err on both sides: outputting a term that does not appear in the private data (false positive), or failing to output a term which does appear. We found it best to use a T that uses only a few samples (for efficiency), with a low threshold to avoid false negatives. To also avoid false positives, we performed further validation before returning S' by running T again on each term. We omitted the validation from Alg 1.

5.2 Optimized Multiple-Term Identification

The MTI algorithm uses the single-term test as a 'black box'. This results in a simple modular algorithm, but also in inefficiencies. One example is that each invocation of the single-term test for a set with y terms, causes the sending of 'dummy' requests for y dummy values; this results in sending the same 'dummy' requests several times to test several sets of terms, instead of using the same 'dummy' requests for testing all of them.

We now describe the main ideas of the OMTI algorithm, an optimized version of the MTI algorithm. OMTI, unlike MTI, tests multiple terms together to minimize overhead. For example, it uses the same 'dummy' requests for multiple tests. OMTI also compares between the times measured for requests with different candidate terms, thereby providing additional useful indication and reducing the errors due to unusually high or low delays to the dummy requests. Namely, even if the times for some set of terms were not significantly higher than those of the dummy requests, but were significantly higher than another set of candidate terms, the set passes the test. These changes improve efficiency and accuracy, allowing a smaller number of requests. To avoid cases in which low measured times for the dummy requests cause all the sets to pass the test, T_θ has a limit on the number of sets it can return.

5.3 Any-Term Identification Algorithm

The ATI algorithm improves accuracy and efficiency for the goal of finding (only) one term in S (that appears in the private data). For simplicity, we present ATI with the assumption that at least one of the terms does appear in the private data; this assumption can be avoided by validating the identified term.

Assuming set S contains at least one term that has search results, facilitating the algorithm's goal to find *any* value in S for which the search query has results, allows ATI to avoid the use of dummy queries. Instead, ATI divides the set S into two subsets S_1 and S_2, compares their loading times against each other, and continues the search with the slower-loading set. ATI continues recursively until one element remains. A simple comparative test suffices to choose the more likely option. We do this by comparing the minimal or average times measured for each of the subsets and taking the subset with the higher value. This relatively simple comparison task allows ATI to be effective, even in cases where distinguishing between two distributions is harder, and hence MTI and OMTI are less effective. See example in [27].

5.4 Property-based Term Identification

The techniques presented in the previous subsections, are impractical if the number of potential terms is huge, e.g., phone numbers, zip codes, credit-card numbers, bank account numbers, or PIN/passwords. However, we can often identify (even) such items, by using special *properties* of the items, such as typical phone-number formats or credit-card-number error detection mechanisms.

Specifically, to identify such items, we usually take advantage of the fact that these complex items are often entered in records using a simple, known structure, often as multiple sets of fixed-length numbers, separated by spaces or other special symbols. We can therefore search each set of digits separately, and then, check for sets that appear in the same message and in 'correct' order. We give two examples: phone numbers and credit card numbers.

Phone numbers. Consider a victim who has her phone number in her email signature, as done by many users. The attacker can launch an effective XS-search attack relying on the Response-inflate technique, as the phone number appears in every sent email. This task is even more efficient since phone numbers have few common formats, and furthermore, they are usually broken into well-defined sets of three or four digits. The attacker can begin with the area code, and continue to the next short sequence.

Credit card numbers. Unlike phone numbers, credit card numbers do not appear many times in Gmail accounts, hence, the Response-inflate technique is less effective here; however, with minor adjustments, the ATI algorithm worked fine, and found credit card numbers in the Gmail accounts of both the authors. Specifically, we use the fact that credit card numbers have fixed structures such as 4-4-4-4. Each 4-digits number has one of $10,000$ options. We slightly modified ATI algorithm: instead of comparing two sets each time, we separated the $10,000$ options into 11 sets (following the length limit of Gmail search request), and instead of continuing to the next round with one of the sets, we continued with 5 of them. We chose to continue with more than 4 options, as we noticed that sometimes the validity year or a wrong number are returned instead of one of the 4-digits values. The output of the algorithm was 5 numbers sorted by their rank. Given the output, we tried to omit the wrong number based on credit card's checksum algorithm. We also examined carefully every two consecutive numbers and numbers that might be the validity year of the credit card. Relying on public information about the prefixes and the structure of credit card numbers, we could reduce the number of options to 2 possible credit card numbers (which can be easily determined; see Section 4.1). Notice that we could use this and additional information about the credit card number structure to improve the efficiency of the attack. See [27] for more details.

For each *sampleSize* $\in \{10, 20\}$, we ran the attack three times on both the accounts, with an average total time of 10.7 and 20.3 minutes respectively. For *sampleSize* $= 10$ we found all the four numbers only once, three of them in three runs, and two in two runs. However, we noticed, that many of the wrong numbers were close to the numbers we missed. For double sample size and time, we succeeded to find the whole credit card number in all the six runs. More details appear in [27].

5.5 Evaluation of Term-Identification Algorithms

In this subsection we validate and evaluate the term-identification attacks using an experiment with Gmail users. In the experiment, we tried to expose the first and the last names of the users (details below). See [27] for additional experiment.

Note that to avoid unnecessary loss of privacy, we limited this experiment to exposing of the user's name, and furthermore, before using our attack to 'guess' the names, we conducted the experiment only with participants who willingly disclosed their names. We evaluated the use of the three algorithms for detecting keywords that appear in many email messages, by identifying the first and last name of the user out of a list of 2000 common names. The attacks used inflating search requests as described in Section 3.3; we used inflating requests based on the results of subsection 3.3.

Participants and process. Participants were 78 (paid, informed and consenting) students, required to have and connect to their (active) Gmail account. Participants were asked to visit the experiment webpage. From this webpage, we launched the algorithms for different small *sampleSize* values: $1, 2$ and 3 only. Namely, when the test T examined a set in MTI, it sent *sampleSize* pairs of challenge (r_C) and dummy (r_D) requests alternated; in OMTI, T sent *sampleSize* requests for each tested set. In the ATI algorithm, only challenge requests were sent.

The challenge search requests (r_C) were for messages in the Sent Mail folder that were sent from one of the names in S. Similarly, the dummy requests (r_D) asked for messages that were sent by one of $|S|$ dummy names that are likely to be unrelated to any Gmail account.

Because ATI returns only one term, we ran it twice to find both the first *and* last names; in the second run we excluded the term identified in the first run. To make a fair comparison with the other algorithms, we ran OMTI's verification step on both the received terms. In the verification step of each of the algorithms, we used *sampleSize* $= 3$, regardless of the *sampleSize* that was used in the regular run.

As a test for MTI and OMTI, we used the minimum test ($T = \mathrm{MIN}_0^{1.05}$). For the OMTI algorithm, to compare between two candidate subsets, we used the test $\mathrm{MIN}_0^{1.1}$. In the ATI algorithm, we simply compared the minimal values of the samples, and continued with the set whose sample had the highest minimal value.

Videos of the attack using each of the algorithms are available online in [28].

We measured the following criteria:

1. *False negative (FN) counter.* For each name that was not identified, we increased this counter by 1.
2. *False positive (FP) counter.* For each name that was returned by the algorithm but was not the name of the participant, we increased this counter by 1.
3. *Total time.* The number of seconds until the test was completed.
4. *Requests counter (RC).* The total number of requests sent during the run.

Table 3 summarizes and compares the results for the three algorithms. The table's columns contain the FN and FP for each configuration, as well as the percentage of runs without any FP and FN (*perfect* runs), the average time of these runs, and the average number of queries sent in them. All the algorithms returned good results. The best values in each category appear in bold and OMTI is almost always best. OMTI had good results (low FN and FP) even for *sampleSize*=1, with 82.1% of the runs being 'perfect' in average time of less than 40 seconds.

The results show that OMTI improves the MTI algorithm both in the quality of the results and in the performance. The only advantage of MTI over OMTI, was in FN rate using *sampleSize*=3; in MTI, only for one participant no name was found (FN counter = 2), compared to 5 participants in OMTI. It seems that the comparison between several candidate sets (ATI compares only 2 sets), is the main reason that OMTI achieved the best results.

Table 3 also contains the results for each of the ATI runs separately (without the final verification step). Their results were impressive, mainly in the time. For example, to find one name (which might be enough in some scenarios), with *sampleSize*=1, we achieved 68% success rate in 12.3 seconds. For double the *sampleSize* and time, ATI achieved an 88.5% success rate.

6. DEFENSES

Server-side defenses. XS-Search attacks use cross-site search requests. Completely blocking cross-site requests would 'break the Web'; defenses should block only cross-site request forgery (CSRF) attacks. While CSRF is often associated with 'causing unwanted action', i.e., *state-changing requests* [29], XS-Search attacks expose information and do not change state or perform actions at the server. Indeed, the sites we tested with search queries, Gmail and Bing, did not apply any of the known anti-CSRF defenses such as anti-CSRF tokens, challenge-response mechanisms (e.g., CAPTCHA), relying on the Referer or the Origin HTTP headers [2], or other defenses (e.g., [11,21,29]).

Both Gmail and Bing prevented users from sending an *excessive number* of search requests within a short time; the limit is about 200 in Bing and 4500 in Gmail. These numbers are too high to disrupt our (efficient) XS-search attacks. A significant reduction in these limits could, of course, reduce the amount of leakage and potentially reduce some legitimate use of cross-site search requests. In particular, websites could require users to re-authenticate or solve CAPTCHA when making unusual cross-site requests (e.g., to history), or when sending too many cross-site queries. Indeed, Google requires users to re-authenticate to perform queries on their search-history log.

Websites can block or limit the effectiveness of Response-inflate XS-search attack, for example by limiting the number of entries returned for a search query or the length of parameters duplicated in the response. It seems more difficult to prevent Compute-inflate XS-search attack, without increasing the response time for benign requests.

Client-side defenses. Several works propose client-side defenses against CSRF attacks. Appropriately designed client-side defenses, such as of [11], can prevent XS-search attacks. These defenses require minimal server-side support; however, all known client-only proposals may not be usable due to false positives. Furthermore, some more advanced client-only defenses such as BEAP [25] would also enable XS-search attacks, since they do not strip persistent cookies.

7. CONCLUSIONS AND FUTURE WORK

We show how even a weak, cross-site attacker, is often able to extract sensitive user data, even from prestige services such as Gmail and Bing, by *manipulating the web service, using side-channels,* to expose data.

Our attacks used classical and customized statistical tests, and tailored search algorithms; further research should explore improvements to both tests and algorithms. Other directions for further research include measuring the prevalence of the problem, designing automated means for detecting and mitigating such weaknesses (beyond Section 6), and investigating social and legal aspects such as liability to damages due to cross-site side-channel weaknesses.

8. ACKNOWLEDGMENTS

We would like to thank Amit Klein, Alexei Czeskis, Hemi Leibowitz, Google Security team and the anonymous reviewers for useful feedbacks. This research was supported by a grant from the Ministry of Science and Technology, Israel.

9. REFERENCES

[1] Apache-Commons. Commons Math: The Apache Commons Mathematics Library . online.

[2] A. Barth, C. Jackson, and J. C. Mitchell. Robust defenses for cross-site request forgery. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 75–88. ACM, 2008.

[3] D. Bleichenbacher. Chosen ciphertext attacks against protocols based on the RSA encryption standard PKCS #1. In *Advances in Cryptology – Crypto 1998*, volume 1462 of *Lecture Notes in Computer Science*, pages 1–12, 1998.

[4] A. Bortz and D. Boneh. Exposing private information by timing web applications. In C. L. Williamson, M. E. Zurko, P. F. Patel-Schneider, and P. J. Shenoy, editors, *WWW*, pages 621–628. ACM, 2007.

[5] C. Bösch, P. H. Hartel, W. Jonker, and A. Peter. A survey of provably secure searchable encryption. *ACM Comput. Surv*, 47(2):18, 2014.

[6] D. Brumley and D. Boneh. Remote timing attacks are practical. *Computer Networks*, 48(5):701–716, 2005.

Algorithm:	MTI			OMTI			ATI twice			First ATI run			Second ATI run		
sampleSize:	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3
FN counter = 1	39.7	17.9	14.1	**11.5**	**5.1**	**2.6**	46.2	9	7.7	32.1	11.5	7.7	43.6	9	7.7
FN counter = 2	21.8	9	**1.3**	**6.4**	**5.1**	6.4	15.4	7.7	5.1	–	–	–	–	–	–
FP counter = 1	9	5.1	7.7	**7.7**	**1.3**	**2.6**	11.5	7.7	6.4	32.1	11.5	7.7	43.6	9	7.7
FP counter = 2	**0**	1.3	**0**	1.3	1.3	**0**	**0**	**0**	1.3	–	–	–	–	–	–
Perfect counter	34.6	69.2	79.5	**82.1**	**89.7**	**91**	38.5	83.3	87.2	67.9	88.5	92.3	56.4	91	92.3
Perfect: time (sec)	52.2	94.2	130.9	39.8	62.5	85.7	**28.2**	**51.4**	**72.4**	12.3	24.2	36.5	12.2	24.2	34.8
Perfect: avg RC	122.1	207.6	282.6	81.6	130.9	181.1	**52.1**	**95.4**	**138.6**	21.6	43.1	64.8	21.6	43.2	64.7

Table 3: Finding two names out of 2000 options by Response-inflate XS-search attack with MTI and OMTI, and by running ATI twice. In bold are the best values per *sampleSize*. The maximal value for the FN and FP counters in some run, was 2.

[7] S. Chen, R. Wang, X. Wang, and K. Zhang. Side-channel leaks in web applications: a reality today, a challenge tomorrow. In *Security and Privacy (SP), 2010 IEEE Symposium on*, pages 191–206. IEEE, 2010.

[8] B. Chor, E. Kushilevitz, O. Goldreich, and M. Sudan. Private information retrieval. *Journal of the ACM (JACM)*, 45(6):965–981, 1998.

[9] J. Clarke. *SQL injection attacks and defense*. Elsevier, 2012.

[10] S. A. Crosby, D. S. Wallach, and R. H. Riedi. Opportunities and limits of remote timing attacks. *ACM Transactions on Information and System Security (TISSEC)*, 12(3):17, 2009.

[11] A. Czeskis, A. Moshchuk, T. Kohno, and H. J. Wang. Lightweight server support for browser-based csrf protection. In *Proceedings of the 22nd international conference on World Wide Web*, pages 273–284. International World Wide Web Conferences Steering Committee, 2013.

[12] K. P. Dyer, S. E. Coull, T. Ristenpart, and T. Shrimpton. Peek-a-boo, I still see you: Why efficient traffic analysis countermeasures fail. In *IEEE Symposium on Security and Privacy*, pages 332–346. IEEE Computer Society, 2012.

[13] C. Evans. Cross-domain search timing. blog, December 2009. http://scarybeastsecurity.blogspot.co.il/2009/12/cross-domain-search-timing.html.

[14] E. W. Felten and M. A. Schneider. Timing attacks on web privacy. In D. Gritzalis, S. Jajodia, and P. Samarati, editors, *ACM Conference on Computer and Communications Security*, pages 25–32. ACM, 2000.

[15] A. Futoransky, D. Saura, and A. Waissbein. The ND2DB attack: Database content extraction using timing attacks on the indexing algorithms. In D. Boneh, T. Garfinkel, and D. Song, editors, *WOOT*. USENIX Association, 2007.

[16] Y. Gilad and A. Herzberg. Spying in the Dark: TCP and Tor Traffic Analysis. In S. Fischer-Hübner and M. Wright, editors, *Privacy Enhancing Technologies Symposium*, volume 7384 of *Lecture Notes in Computer Science*, pages 100–119. Springer, 2012.

[17] O. Goldreich and R. Ostrovsky. Software protection and simulation on oblivious rams. *Journal of the ACM (JACM)*, 43(3):431–473, 1996.

[18] Google. Advanced search. `https://support.google.com/mail/answer/7190?hl=en`.

[19] Google. Standard view and basic html view. `https://support.google.com/mail/answer/15049?ctx=gmail`.

[20] M. Johns, S. Lekies, and B. Stock. Eradicating DNS rebinding with the extended Same-Origin Policy. In *USENIX Security*, pages 621–636, 2013.

[21] N. Jovanovic, E. Kirda, and C. Kruegel. Preventing cross site request forgery attacks. In *Securecomm and Workshops, 2006*, pages 1–10. IEEE, 2006.

[22] Ç. K. Koç. About cryptographic engineering. In Ç. K. Koç, editor, *Cryptographic Engineering*, pages 1–4. Springer, 2009.

[23] P. C. Kocher. Timing Attacks on Implementations of Diffie-Hellman, RSA, DSS, and Other Systems. In N. Koblitz, editor, *CRYPTO'96*, volume 1109 of *LNCS*, pages 104–113. IACR, Springer-Verlag, Germany, 1996.

[24] E. L. Lehmann and J. P. Romano. *Testing statistical hypotheses*. springer, 2006.

[25] Z. Mao, N. Li, and I. Molloy. Defeating cross-site request forgery attacks with browser-enforced authenticity protection. In *Financial Cryptography and Data Security*, pages 238–255. Springer, 2009.

[26] Microsoft. Internet Explorer Dev Center - Timing and Performance APIs. `http://msdn.microsoft.com/en-us/library/ie/hh772738(v=vs.85).aspx`.

[27] Nethanel Gelernter and Amir Herzberg . Cross-Site Search Attacks, technical report 15-01. `http://u.cs.biu.ac.il/~herzbea/security/15-01-XSSearch.pdf`, August 2015.

[28] Nethanel Gelernter and Amir Herzberg. Demo: XS-Search attack on Gmail, May 2015. Online at `http://u.cs.biu.ac.il/~gelernn/xssearch/`.

[29] Paul Petefish, Eric Sheridan, and Dave Wichers. Cross-Site Request Forgery (CSRF) Prevention Cheat Sheet. `https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet`.

[30] A. Shamir. A top view of side channel attacks. In *Proc. of L-SEC/CALIT IT Security Congress (October 19-20, 2006)*, 2011.

[31] Z. Weinberg, E. Y. Chen, P. R. Jayaraman, and C. Jackson. I Still Know What You Visited Last Summer: Leaking Browsing History via User Interaction and Side Channel Attacks. In *IEEE Symposium on Security and Privacy*, pages 147–161. IEEE Computer Society, 2011.