Google Hacking Project
BadSense — Using Google's advertising mechanisms as a hacking tool

Ohad Lutzky, Tom Meiri, Guy Treger
Under the supervision and guidance of
Amichai Shulman

Contents

Part I
Introductory Details

1 Introduction — Google and Hacking

Google has been a well-known secondary utility in website attacks for quite a while, used as a probe for vulnerable sites or leaked information. This usage relies on constructing queries for specific strings which are known to appear on vulnerable sites (e.g. version numbers). Google's enormous site database and advanced search modifier keywords (such as `inurl` or `filetype`) allow for particularily elaborate probing techniques. There are existing libraries of malicious queries[1].

1.1 Active attacks using Google

Our objective is to use Google as an *active* part of an attack. This gives us several advantages over attacking directly; an immediate benefit is a layer of anonymity, granted by any attack-by-proxy. Another benefit granted by this is abuse of the implicit trust of Google by webmasters; Google is unlikely to be blocked from entering a website, which allows an attacker to attack a website even if his IP address is blocked from accessing the website. Some sites may even allow Google's bots to enter protected segments of the website, for indexing purposes.

1.1.1 Shortcomings of attacking with GoogleBot

In our experimentation, we found that for low-ranked websites, Google's primary indexing robot (GoogleBot) enters at a very low frequency, and inter-visit times may reach several weeks. The frequency remains low even when requesting a re-index with Google's webmaster tools. This makes it difficult to utilize for active attacks. Also, the terms which are likely to show up on Google searches in passive attacks are limited to those resulting from entering legitimate (i.e. appearing as links on highly-ranked websites) URLs, decreasing the amount of information which can be gained. These properties make MediaBot (described in 2.2) better-suited for our attacks.

2 Google's Advertising Mechanisms

Google uses two primary mechanisms for its advertising business front-end, known as AdWords and AdSense.

2.1 AdWords

AdWords allows customers to buy ads to be displayed (on sites with AdSense, described later). Bids can be placed either by CPM (cost per thousand impressions) or by CPC (cost per click)[2]. The bids are either on keywords which would trigger the ad's appearance, or on URLs on which the ad should appear.

An ad contains a title, two lines of text, a display URL and a link target (perhaps different from the display URL). Google has rather strict limitations on what kind of text is allowed, within the ad and at the link's target website, but they are not immediately (or, it seems, strongly) enforced.

2.2 AdSense

AdSense allows webmasters to place an ad "billboard" on their site, sharing Google's revenue per click or impression. When a user enters a site which contains AdSense ads, the browser runs a script which requests a relevant ad for the page. At this point, Google *immediately* sends MediaBot to scan the

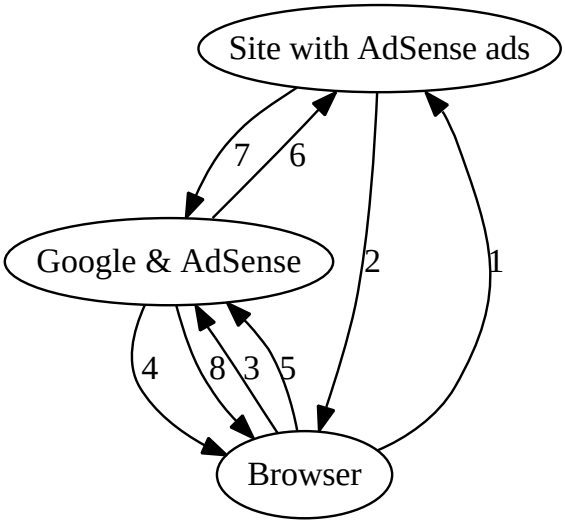

Figure 1: Illustration of the AdSense process

reported "current page" for relevant keywords, as part of the ad selection process. The AdWords database is scanned for an appropriate ad, and it is displayed within the website. The process is detailed below (and illustrated in Figure 1).

1. Browser requests page from website.
2. Website returns page HTML; include references to AdSense script, in a form similar to this:

```
<script type="text/javascript"><!--
google_ad_client = 'pub-0123456789012345';
/* 125x125 created 1/2/03 for Israel Israeli */
google_ad_slow = '9876543210';
google_ad_width = 125;
google_ad_height = 125;
//-->
</script>
<script>
<script type="text/javascript"
src="http://pagead2.googlesyndication.com/pagead/show_ads.js">
</script>
```

3. Browser request AdSense script from Google.
4. Google sends AdSense script to the browser, which runs it.
5. Script reports current page, requesting an appropriate ad from Google.
6. MediaBot sends the site a request for the reported page.
7. Site returns the requested page; Google scans it for Keywords.
8. Google picks an ad and sends it to the browser.

3 Additional technical details

3.1 Forms of HTTP requests

There are two primary forms of HTTP requests — GET and POST. GET requests are the simpler form of HTTP communication; all parameters, if any, are passed as part of the URL. This is the ordinary request sent when visiting a page with a browser. Parameters are encoded in the URL in the following form:

`http://`*url*`?<`*param*1`>=<`*value*`>&<`*param*2`>=<`*value*`>&...`

For example,

`http://www.google.com/search?q=differential+cryptanalysis%27&btnG=Search`

In the instance above, two parameters are passed as part of the URL: first, the parameter `q` is set to "`differential cryptanalysis'`"; spaces are encoded as `+`, and `%27` is the hexadecimal encoding of ASCII character 0x27, which is a quote. The parameter `btnG` is set to "`Search`".

HTTP POST requests can have additional parameters passed in the request body. These requests are usually generated by web forms. They are also considered by the HTTP standard to be the request form for any kind of request that modifies the website, or performs any sort of active or potentially destructive server-side operation. For this reason, pre-fetching browsers do not follow any web forms which perform POST requests, and all browsers give a warning when re-submitting a POST request due to a click on the "Back" or "Refresh" buttons.

While it is considered "best practice" to place all potentially destructive operations behind a POST request, many sites do not follow these practices, and for our demonstration we'll assume the victim does not.

3.2 SQL Injections

Some sites are naive when passing user data to SQL. For example, when authentication a user, the site might use code similar to this:

```
query("SELECT COUNT(*) FROM users WHERE user='" +
$POST['user'] + "' AND pass='" + $POST['password'] + "'")
```

Note that if the user's name is "`Robert'); DROP TABLE Students;--`"[3], then the following query will be run:

```
SELECT COUNT(*) FROM users WHERE user='Robert'; DROP TABLE Students;--' and PASS=...
```

The query is truncated here, as `--` is the comment-to-end-of-line character for some common SQL dialects.

As some sites have SQL injections in their search pages (which are a legitimate case of a GET request), this allows an external user to craft a malicious GET request to attack them.

4 Methodology and failed attempts

For the course of the project, we've set up several AdSense accounts, as well as several anonymous sites on free hosting services which included support for scripting using PHP. This allowed us to write pages with access logs, and analyze the various bot behaviours.

Initially we attempted to attack the AdSense Site Authentication mechanism; this feature allows a webmaster to provide a username and password (and login URL) which would be used by MediaBot when crawling the site. The plan was to get MediaBot to log into one site using credentials for another. However, at the time we could not get this feature to work at all — MediaBot would not visit the

login page, and the only effect of activating Site Authentication was that MediaBot would not visit the portions of the site we told it were restricted. It seems that this feature does not work at all anymore[6]. It is possible that a vulnerability was indeed discovered, causing Google to disable the feature.

An additional attack we tried used the "Allowed Sites" AdSense feature. After adding a site to our "Allowed Sites" list, we placed text that violates the terms-of-service for AdSense (such as "please click the ad") on it. However, the control for this experiment failed as well, as we received no notice from Google when a site directly violated the terms of its own AdSense account in this method.

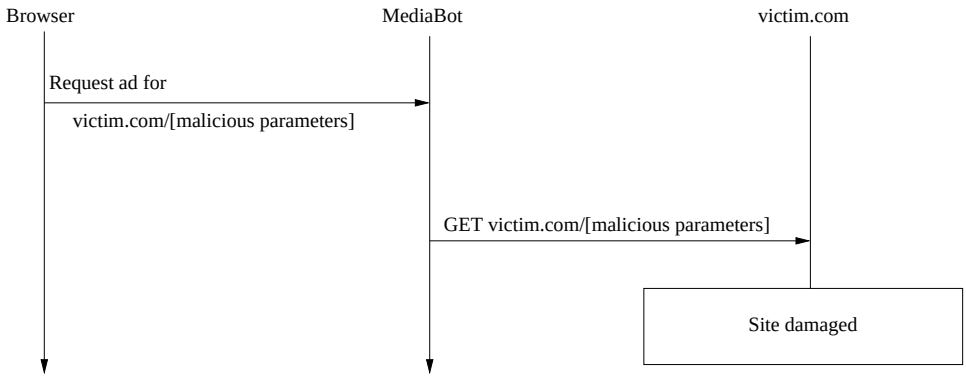

Figure 2: AdSense-Out

Part II

Our attacks

We present two attacks based on AdSense, and one attack based on AdWords. The first AdSense attack is able to cause damage to a website, and is therefore called "AdSense-Out". The second AdSense attack is able to retrieve information from a website, and is therefore called "AdSense-In". For completeless, the third attack is known as "AdWords-Hokey-Pokey".

5 AdSense-Out

As mentioned in 2.2, the user's browser asks Google for an ad to be displayed for the current page. The AdSense script uses JavaScript's `document.location` to emit the current URL[4]. However, no verification of this URL is performed, and at the present day it is possible to craft an AdSense request requesting an ad for any URL, simply by modifying the GET parameter "`url`". An analysis of the (compressed and obfuscated) AdSense code yielded a simple method to craft these requests: The `google_page_url` parameter can be set in the JavaScript preamble of the AdSense code to achieve the same effect.

Testing shows that there is no a priori verification of which pages may be reported as the "current page". There is no filtering of GET parameters, probably due to the fact that ads can be shown on internal parts of dynamic sites, such as online forums. While there is a limit on the length of the reported URL, MediaBot appears to follow HTTP redirects, which allows one to use online services[5] to get a short URL which would redirect MediaBot to a longer one.

An attack is made possible, therefore, by sending MediaBot to a URL which, just by visiting it, would trigger an SQL injection attack, or pass otherwise malicious parameters, as shown in Figure 2. This has been tested by performing an SQL injection attack, defacing a specially-prepared site which is vulnerable to SQL injections.

6 AdSense-In

In contrast with direct attacks on websites, attacks using an intermediate proxy have the disadvantage of receiving no data from the site, as the response is sent to the proxy. A direct attack might yield plentiful information: Success status of the attack is usually immediately discernable from the response. Furthermore, additional hidden information can be obtained using SQL's `UNION` modifier. For example, consider a site which allows searching for products using a search string in the following implementation:

```
query("SELECT name,description FROM products WHERE name='" + $GET["name"] + "'")
```

Then an attacker might search for a product with the following name:

```
' AND 1=0 UNION SELECT name, creditcard FROM customers --
```

This would cause no products to be displayed (as the `1=0` clause is always false), but instead all customers will be displayed as products, with the product description being their credit card number.

6.1 Receiving information from an AdSense attack

The AdSense process, as detailed in 2.2, has two additional steps, which allow us to retrieve information. This channel is limited, as the only returned information is contextual ads. Our strategy will therefore be to modify the result page in such a way that the chosen ads will pass information to the attacker.

The simplest solution is a binary one — we will "plant" an ad topic in the returned page in case of a successful attack. Alternatively, in case we already know the attack will succeed, we can plant the topic in the returned page if a binary SQL query of our choice returns "true".

For example, in order to perform a `UNION` attack, we need to know how many fields the original query has, otherwise an SQL error will be returned. In a direct attack, one might use a query similar to this:

```
http://victim.com/search?q=' AND 1=0 UNION SELECT null,null,null --
```

If the original query has 3 fields, one result (with empty fields) will be shown, and otherwise an SQL error will be displayed (complaining that the number of fields does not match). In an attack using AdSense, we can replace one of the fields (which hopefully expects a string data-type) to contain keywords of our choice. For example, one might use a query similar to this:

```
http://victim.com/search?q=' AND 1=0 UNION
SELECT 'car automobile mechanic clutch muffler exhaust',null,null --
```

For this query, one might expect that a correct guess (3 fields) would cause a single entry with the text "car automobile mechanic clutch muffler exhaust" to appear in the response, causing the chosen ad to be car-related. An incorrect guess would not cause this entry to be shown. However, MediaBot extracts keywords from the URL itself, causing a car-related ad to be shown regardless of the success of the query. We will show how to overcome this in 6.1.1.

We can use a more elaborate version of this technique to systematically extract all usernames from a system:

```
http://victim.com/search?q=' AND 1=0 UNION
SELECT 'car automobile mechanic clutch muffler exhaust',null,null
FROM customers WHERE lastname LIKE 'A%' --
```

This query will return a car-related ad if the site has any customers whose last name begins with an A. These queries can be repeated and extended in order to retrieve any information from the site (which is attainable by SQL injection) by means of binary search.

As these queries become quite long, the attacker will need to use a redirection service as described earlier to generate short URLs for the attacks.

6.1.1 Using SQL `CAST` to hide keywords in the URL

In order to prevent the keywords from appearing in the URL of the request, we need to encode them in a way which will be decoded by the SQL server. One way to do this is with SQL's `CAST` statement. For example, this query

```
http://victim.com/search?q=' AND 1=0 UNION
SELECT 'car automobile mechanic clutch muffler exhaust',null,null --
```

Can be written as

```
http://victim.com/search?q=' AND 1=0 UNION SELECT
CAST(0x6300610072002000610075007400 6f006d
006f00620069006c00650020006d0065006300680
```

```
061006e006900630020006300 6c00750074006300
680020006d007500660066006c006500720020006
500780068006100750073007400 AS
NVARCHAR(1000)),null,null --
```

The URL contains no trace of the keywords, as far as MediaBot can tell, but the query is completely equivalent. The long hexadecimal number is simply a little-endian word-per-character ASCII representation of the string.

6.1.2 If at first you don't succeed

In some instances, AdSense does not immediately return a relevant ad for a fresh URL. A new URL should be generated in this case, as the old URL would return a cached result. Therefore one should use URLs such as:

```
http://victim.com/search?trash=1234&q=' AND 1=0 UNION SELECT
CAST(0x6300610072002000610075007400 6f006d
...
500780068006100750073007400 AS
NVARCHAR(1000)),null,null --
```

Replacing the `trash` value at every attempt in order to create a fresh URL.

7 AdWords-Hokey-Pokey

AdWords-Hokey-Pokey is a method to create more efficient Cross-Site Request Forgery (CSRF) attacks.

7.1 Cross-Site Request Forgery attacks

Many potentially destructive operations within sites can only be performed by a logged-in user, on his own account. Many browsers today have shared sessions between multiple instances of the browser, be they separate windows or tabs[1]. An attacker can use this fact in order to conduct a CSRF attack: Assuming that the victim user is already logged into a certain victim site, and also visits the attacker's site with the same browser, the attacker can cause the browser to unknowingly execute business transactions in the victim site.

One disadvantage of this technique is that it requires guesswork — the attacker cannot usually know whether the victim user is already logged into the victim site.

7.2 Using AdWords for CSRF attacks

As AdWords allows advertisers to select specific URLs to advertise on, we can attack by advertising a malicious URL, requesting that the ad be shown on a page which suggests that the users is logged into the victim site. For this to work, there has to be such a page which displays AdSense ads. Furthermore, the weakness of the victim site has to be exploitable via GET requests in order for the ad's link to attack directly.

An alternative is to link to a secondary site which performs the attack. However, this might not allow POST-based attacks, as modern browsers defend against cross-site JavaScript requests. The secondary site can still exploit a GET vulnerability using, for example, an `iframe`. This has the advantage of the attacker being able to quickly modify the site, as opposed to the AdWords ad, which would take several hours to update. However, this will cause the Referer field of the malicious GET request to reveal the attacking website, whereas a direct malicious ad-link will show Google as the attacker.

[1] Even multi-process browsers such as Google Chrome have this property

Two additional issues to consider with the direct ad-link attack pertain to the URL of the ad. The URL length is limited, in this instance, to 1024 characters, which may be insufficient. Also, while the display URL can be chosen at will, the actual link does show that the ad leads to the same site the user is currently on. Yet the link goes through a Google Syndication redirect page, and the actual target URL parameter is preceded by a long preamble, which is likely to prevent it from being shown in the status bar while hovering over the link; the user would have to right-click the ad link and examine it before noticing the suspicious link.

8 Concluding remarks

We have shown that Google's advertising mechanism can be used as an *active* attack tool. Additionally, we have shown a method to cause websites to leak sensitive information via contextual advertising services. It is imperative to take Google utilities into account when considering the security of a website.

Further research is required to determine whether other Google utilities can be used for active attacks. Also, the question arises whether it is Google's responsibility to protect against these attacks.

References

[1] Johnny.IHackStuff.com, http://johnny.ihackstuff.com

[2] AdWords Help: "*How do placement-targeted and keyword-targeted ads compete?*", http://adwords.google.com/support/bin/answer.py?hl=en&answer=18280

[3] xkcd, *Exploits of a Mom*, http://xkcd.com/327

[4] The AdSense script, http://pagead2.googlesyndication.com/pagead/show_ads.js

[5] TinyURL http://tinyurl.com

[6] AdSense Help Google Group: "*AdSense Site Authentication on forums*" http://groups.google.com/group/adsense-help-troubleshooting/browse_thread/thread/60c31bc21eaf3997

List of Figures