Start Secure. Stay Secure.™

SPI Labs Research Brief

Stealing Search Engine Queries with JavaScript

By SPI Labs

Overview

SPI Labs has expanded on existing techniques and discovered a practical method of using JavaScript to detect the search queries a user has entered into arbitrary search engines. As seen with the recent leakage of 36 million search queries made by half a million American Online subscribers, there are enormous privacy concerns when a user's search queries are made public. All the code needed to steal a user's search queries is written in JavaScript and uses Cascading Style Sheets (CSS). This code could be embedded into any website either by the website owner or by a malicious third party through a Cross-site Scripting (XSS) attack. There it would harvest information about every visitor to that site. For example, an HMO's website could check if a visitor has been searching other sites about cancer, cancer treatments, or drug rehab centers. Advertising networks could gather information about which topics someone is interested based on their search history and use that to echance their customer databases. Government websites could see if a visitor has been searching for bomb-making instructions.

JavaScript URL Detection

As originally presented by Jeremiah Grossman at Black Hat USA 2006 [1], JavaScript can be used with CSS to detect if someone has visited an arbitrary URL. This works because CSS can define the styles for visited and non-visited hyperlinks. Since JavaScript can access the style of any element, it can see which style the browser has applied and determine whether a user has visited the link. It is interesting to note that the CSS standard warns that a malicious website could abuse CSS to see which sites a user has visited [2]. Also important is that URL detection technique cannot be used to enumerate through a user's browsing history. JavaScript and CSS can only be used to see if a user visited a specific URL. This means JavaScript would not know that someone had visited *spidynamics.com* unless the script explicitly checks for that domain. As long as someone knows which specific URLs to check for, URL detection can be extended to steal more private information than just a user's browser history.

Search Engine Results Pages

Typical search engines have a web form that users type their search queries into. When the user submits this form, an HTTP GET request is made to the search engine and a page containing the search results for the supplied query is returned. It is important to note that there is nothing special about the search engine's web form, and simply calling the search engine's results page directly with a search query will return the same results. The URL for various search results pages tends to be nearly identical regardless of the search query. As you can see in Figure 1, the only difference is the query a user was searching for.

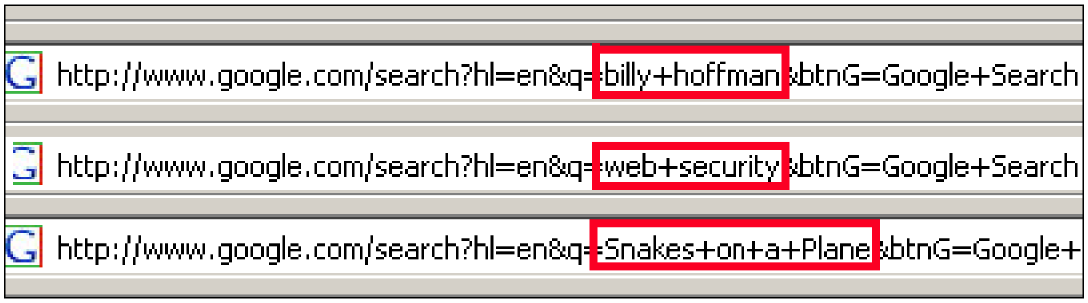

http://www.google.com/search?hl=en&q=billy+hoffman&btnG=Google+Search
http://www.google.com/search?hl=en&q=web+security&btnG=Google+Search
http://www.google.com/search?hl=en&q=Snakes+on+a+Plane&btnG=Google+

***Figure 1:** URLs of Google's results page for different search queries*

JavaScript can be used to determine whether a user has visited the URL of the results page for a given query and thus determine if a user has searched for that query.

Dynamic Nature of Search Engine URLs

There are numerous issues with the above approach that has led some people to believe that it is not practical for an attacker to steal search queries using JavaScript. First, the query string of the URL for a search engine's results page can change based on how the user performed a search. For example, if a user typed "web security" into Google's main page and pressed the enter key inside the text box to submit the query, the URL of search results page would be */search?hl=en&q=web+security*. However, if a user instead clicked the "Google Search" button to submit the query, the URL of the search results page would be */search?hl=en&q=web+security&btnG=Google+Search*. If a user were to enter a search query into the web form that is at the top of Google's results page as opposed to the web form on Google's main page the query string of the results page's URL would be different yet again. Figure 2 illustrates the differences among these URLs.

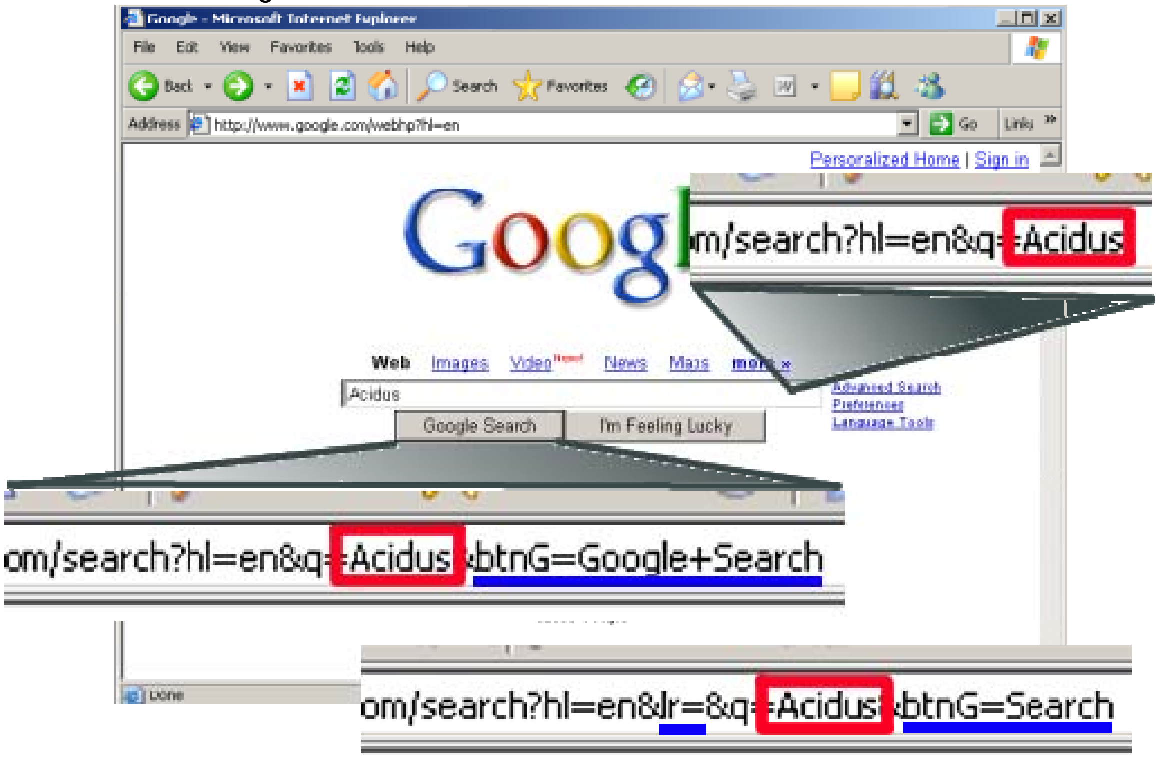
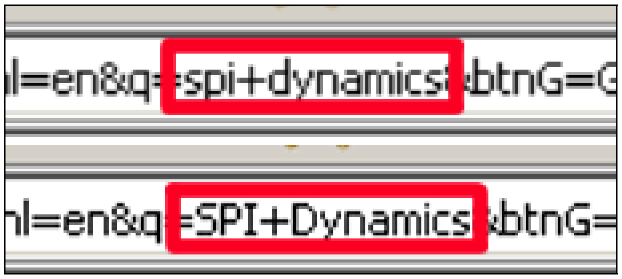

Google - Microsoft Internet Explorer
http://www.google.com/webhp?hl=en
Acidus
Google Search
I'm Feeling Lucky
m/search?hl=en&q=Acidus
om/search?hl=en&q=Acidus&btnG=Google+Search
om/search?hl=en&lr=&q=Acidus&btnG=Search

Figure 2: *The differences in the query strings (underlined in blue) of the URLs for Google's results pages using the same search query (highlighted in red).*

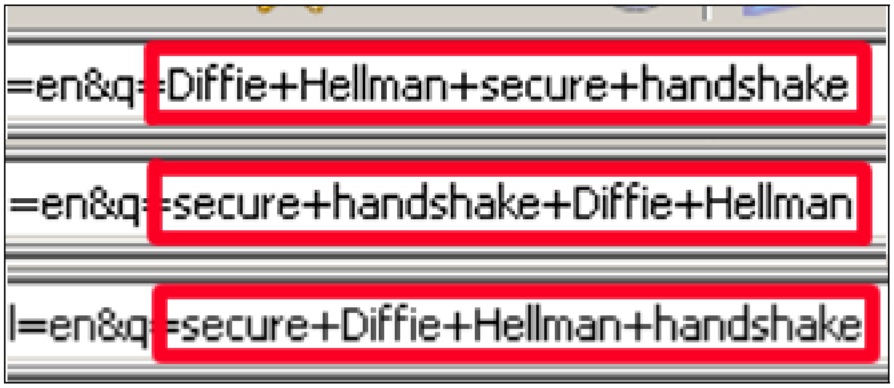
Second, the letter case of a search query also matters. As Figure 3 shows, queries with the same words that use different letter casing will result in two different URLs for the results page. For queries consisting of multiple words, each distinct ordering of these words also results in different URLs for the results page as shown in Figure 4.



l=en&q=spi+dynamics&btnG=G

hl=en&q=SPI+Dynamics&btnG=

Figure 3: *Different URLs for different letter casing*



=en&q=Diffie+Hellman+secure+handshake

=en&q=secure+handshake+Diffie+Hellman

l=en&q=secure+Diffie+Hellman+handshake

Figure 4: *Different URLs resulting from different word orderings*

Overcoming Dynamic URLS for Results

Since many factors can cause different URLs for very similar search queries, it is helpful to see how these factors affect the practicality of detecting a user's search engine queries using JavaScript URL detection. Consider using JavaScript to see if a user has searched Google for some variation of "Diffie Hellman key exchange." Given *x* words, there are 2^x possible combinations where the first letter of each word is upper or lowercased. There are also *x!* possible ways to order *x* words. Each possible search query must be checked against the different URLs for the results page that varies on how the user performed the search. With Google, we will only check the 3 URLs associated with searches from the main page or from a search results page. Using these figures, there are $(2^4 * 4!) * 3 = 1152$ different URLs that return results for some variation of “Diffie Hellman key exchange.”

Remember, JavaScript can be used to determine if a user visited a link very quickly, allowing for thousands of links to be checked a second. Generating all the possible combinations for a multiple word search query in an interpreted language like JavaScript is also viable. In tests on a 3Ghz desktop, SPI Labs was able to generate the 384 (24 * 4!) variations of a 4 word search query in half a second. All variations of a 6 word search query, some 46,080 combinations, were generated in 5 seconds. This means that using JavaScript to generate all possible mutations of a given query string and check each one is practical given the power of modern computers.

“What Do You Want for Christmas?”

The concepts behind URL detection and the code to generate all possible search query mutations are easy to grasp and implement. However, this merely provides the infrastructure to detect whether a user has searched for some variation of a given query. It does help someone decide which search queries to look for.

An entity may already know which queries to detect. For example, *fbi.gov* could push JavaScript to see if a visitor to its site has searched for keywords associated with child pornography. JavaScript could be used to automatically submit a post to the FBI tips page with the visitor's internal and external IP addresses as well as the terms used in the search. Consider a fictitious online books store, *spibooks.com*. When a user searches *spibooks.com* for books on web security, *spibooks.com* could push down JavaScript to see if that user has already looked for similar titles on *amazon.com* or another online bookstore. *spibooks.com* could also determine whether the user has searched another online bookstore for terms similar to the search terms they submitted to *spibooks.com* to learn more about that user.

There are applications for these techniques in online advertising as well. An advertiser could leverage URL detection techniques to determine whether a user has visited certain bell weather sites for a given topic, such as Ars Technica, Slashdot, or Microsoft's MSDN. Then JavaScript could use on-demand scripting to receive a list of specific queries based on which topics the user seems to enjoy. All of this information could be aggregated using web analytics software to provide information on the types of products people researched before purchasing, ratio of informed customers to uninformed customers, how informed a customer is that does make a purchase, and so on.

Finally, there are some interesting problems in how to effectively generate queries on the fly instead of using a precompiled list. JavaScript could be supplied a list of popular adjectives, nouns, verbs, and topical proper nouns.

Queries could be assembled from this list of words like Mad Libs™. JavaScript could randomly select from the appropriate list and fill in format strings such as "[adjective] [adjective] [noun]" or "[proper noun] [verb] [noun]" to try to guess search queries.

Proof of Concept Code

SPI Labs has created SearchTheft as a proof of concept of the above techniques. It is written in JavaScript and has been tested against both Mozilla/Firefox and Internet Explorer. SearchTheft automatically generates all letter casing and word order permutations of a given search query and checks if the user has searched for some variation of that query on a wide variety of search engines. Adding support for a new search engine is as simple as entering a few URLs into the SearchTheft's *loadEngines()* function. SearchTheft will only check the search engines that the user actually visits so there is no downside to adding multiple search engines.

A demonstration of using SearchTheft in a web page is available to the public, and can be found at http://www.spidynamics.com/spilabs/js-search/. SearchTheft does not report any of its findings back to SPI Dynamics. Instructions are included on the Web page.

Disclosure of Information

The ability to steal a user's private search engine queries and expose them to third parties drastically increases the damage XSS can do. SPI Labs is releasing this information to publicly demonstrate just how damaging JavaScript can be and to ensure that as many people as possible are aware of this increased danger from XSS vulnerabilities.

Recommendations

To protect themselves from this threat end users should routinely clear their browser's history.

- **For Internet Explorer:** Go to Tools->Internet Options->General and click the "Clear History" button. Users can also configure Internet Explorer to automatically clear its browsing history.
- **For FireFox:** Go to Tools->Options->Privacy and click the "Clear Browsing History Now" button. Users can also click the "Settings" button here and configure Firefox to clear its browsing history every time Firefox is closed.

Developers can reduce the risk of exposing the privacy of their users by securing their websites against XSS using the following recommendations:

- Ensure that all input is validated before being processed.

- Use whitelisting rather than blacklisting for input validation. Whitelisting involves only accepting what you know to be good data, while blacklisting uses a list of data not to allow. Looking for known, valid, and safe input is much easier than looking for potentially unknown dangerous input. For example, you know that a zip code should always be five numbers; whitelisting a zip code input means accepting only five numbers and nothing else.
- Have your applications assessed for security vulnerabilities throughout the software development lifecycle.

A general rule of thumb is: "Never trust user." In most cases, attempting to remove dangerous meta-characters from the input stream leaves a number of risks unaddressed. Developers should restrict variables used in the construction of pages to those characters that are explicitly allowed (similar to firewall rules where you begin with "deny all" and then open only certain ports).

Additional Information

Additional information on possible Cross-Site Scripting attacks can be found at the following locations:

Cross-Site Scripting: Are your Web applications vulnerable?
http://www.spidynamics.com/whitepapers/SPIcross-sitescripting.pdf

The Cross-Site Scripting FAQ

http://www.cgisecurity.com/articles/xss-faq.shtml

References

1. http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Grossman.pdf
2. http://www.w3.org/TR/CSS21/selector.html#link-pseudo-classes)