

Sunday, January 10, 2010

### SQLi Detection - Duh Moment

Not sure why it took me so long to figure out what I am sure is obvious to most other people who have thought about this, but it never clicked for me how to get anywhere near useful SQL Injection detection. The injection itself is trivial, of course, but determining whether it actually worked and weeding out false positives in an automated manner was something that seemed too hard.

During my run on Friday I had a Duh! moment on it. Annoyingly simple. Do it in 3 requests. Request #1 is a normal request. For example, "?id=1" in the URL. If the id is being passed to an SQL request it will return a single record or perhaps no record, it doesn't really matter. Now on request #2 do "?id=1 or 3=4", that is, inject a false 'OR' condition. If the output changes, we are done. Nothing to see here. However, if the output does not change we send request #3 with "?id=1 or 3=3" and if that output differs from request #2 then we have a potential SQLi situation. There are of course still chances of false positives (and negatives) with page stamps and such, but filtering out the response headers and html comments cuts down on that a bit. Add different combinations of single and double-quotes, like "?id=1'or'3='3" (without the double-quotes, of course) and it might be able to catch something.

The best thing about it is that it can slide into an existing scanner framework quite easily. If you have a base reference request, then it just adds a single request to the common case where the false 'OR' condition output does not match the base reference. You only need to do the true 'OR' condition request in case it does match.

Anybody have any other approaches?

Posted by Rasmus in Software at 18:44

Rad, I take the 'encode everything' approach, and only decode explicitly when I need to.  
Anonymous on Jan 10 2010, 20:04

My own approach is quite easy too.

First request is "id=0", then "id=0 or 1". If I get more (all) results in the second time while id=0 returns nothing, I'm done.

I found this recently in a web counter service, where the customer id can be changed in the image url. Some playing around made me (probably) add one visit to everyone's counter.

Anonymous on Jan 10 2010, 21:06

Right, but that results in a lot of false positives. For example, for a search page searching for "0" is likely to give you different results from searching for "0 or 1" and that doesn't mean you have found an SQLi issue. That's why you need the 3 requests:

id=0  
id=0 or 0

if those w give you the same result then you can try:

id=0 or 1

and if that gives you different results from the previous ones, then you might have something.

Anonymous on Jan 10 2010, 21:11

Whenever I find a potential SQLi vulnerable website, more often than it should be, I run a serie of tests - no list here, just intuition - and I can get it to work fine, from simple lists to "union select" or "into file".

The problem is, as you said, automating those tests by eliminating false positives/negatives, and your "3 steps" approach looks simple and fast - going to script it now :)

Anonymous on Jan 11 2010, 01:40

I nearly always use "LIMIT 1" when asking for just one entry to prevent damage etc. ... so this won't work?

Anonymous on Jan 11 2010, 09:58

As an SQLi prevention technique, no, LIMIT 1 isn't going to make you safe. You need to prevent the injection in the first place so the bad guys can't do weird things to your query.

Anonymous on Jan 11 2010, 11:07

## Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Actually, the LIMIT 1 will make it harder for this injection detection to work.  
So in a sense, "this won't work", indeed.  
Anonymous on Jan 13 2010, 08:31

Well, the detection will only fail on the LIMIT 1 if you are really unlucky and you happen to be dealing with the first record. In my actual tool I am appending some numbers to the record id in order to not get any returned, so the "or 3=3" even if there is a limit 1 in there will cause it to return a record and the detection will work fine. The bigger problem with this approach is the low precedence of the OR operator which will cause it to not work at all in more complex SQL queries.  
Anonymous on Jan 13 2010, 08:41