

Blind SQL injection

Optimization techniques

Rodrigo Marcos

December 2007

Agenda

- What is (blind) SQL injection
- Available Open Source tools
- Blind SQL injection optimizations
- Demo
- Conclusions
- Questions

What is SQL injection?

- Well known and exploited technique
- Mainly exploited in web environments
- It abuses improper user input validation
- Allows an attacker to reach the database

SQL injection example

- `http://victim/listproducts.asp?cat=books`
- `SELECT * from PRODUCTS WHERE category='books'`
- `http://victim/listproducts.asp?cat=books' or '1'='1`
- `SELECT * from PRODUCTS WHERE category='books' or '1'='1'`
- Basically, on SQL injection the attacker gets results.

What is blind SQL injection?

- Same vulnerability as SQL injection
- **Very** common vulnerability
- Sometimes (wrongly) ignored during tests as unexploitable or not detected
- The attacker can not retrieve results
- The attacker can only retrieve a True/False condition

Blind SQL injection example

- <http://victim/showproduct.asp?id=238>
- `SELECT * from PRODUCTS WHERE id=238`
- Sometimes, due to the code surrounding the SQL query (grouped or sorted) the attacker can't UNION and no 'good' ways of exploitation are found
- <http://victim/showproduct.asp?id=238 and 1=1>
- <http://victim/showproduct.asp?id=238 and 1=2>
- `SELECT * from PRODUCTS WHERE id=238 and 1=1`
- `SELECT * from PRODUCTS WHERE id=238 and 1=2`
- Blind SQL happens if the requests above return different results

Exploiting True/False conditions

- Select user returns 'dbo'
- **SUBSTRING('Select user', 1, 1) = 'd'**
- **SUBSTRING('Select user', 2, 1) = 'b'**
- **SUBSTRING('Select user', 3, 1) = 'o'**

- **http://victim/showproduct.asp?id=238 and SUBSTRING('Select user', 1, 1) = 'd' → TRUE**
- **http://victim/showproduct.asp?id=238 and SUBSTRING('Select user', 1, 1) = 'X' → FALSE**

Available solutions

- Custom Script: We can script it and discover each letter
 - Set a space: [a-z] + [A-Z] + [0-9] + [symbols]
 - Loop for every character
- Absinthe
 - <http://www.0x90.org/releases/absinthe/>
- BSQLBF
 - <http://www.unsec.net/download/bsqlbf.pl>
 - <http://www.unsec.net/download/bsqlbf.avi>
- SQLMap, SQLBrute.py

Available solutions

- Custom scripts: Not reusable. I got sick of writing dirty BSQL injection scripts...
- Available open source tools:
 - Some of them are too dumb trying to be smart (and don't work in special situations)
 - Most are not interactive
 - None are optimized for speed

Blind SQL injections optimizations

- Narrow down the charset:
 - ASCII(**UPPER**(SUBSTRING((SQL Query), Position, 1)))= ASCII(SUBSTRING((SQL Query), Position, 1))
 - ASCII(**LOWER**(SUBSTRING((SQL Query), Position, 1)))= ASCII(SUBSTRING((SQL Query), Position, 1))
 - If first one true, character is uppercase
 - If second one true, character is lowercase
 - If both are true, it is a number or symbol

Blind SQL injections optimizations

- Searching character space:
- [a,b,c,d,e,f,g,h,i,j,k,l,m,n,o,p,q,r,s,t,u,v,w,x,y,z]
- Sequential search: Not optimum
- Divide and conquer!
 - Character => 'm'
 - [m,n,o,p,q,r,s,t,u,v,w,x,y,z]
 - else
 - [a,b,c,d,e,f,g,h,i,j,k,l]

Blind SQL injections optimizations

- Big numbers for table enumeration
- Typical MSSQL table id: 2089058478
- Absinthe:
 - Increase exponentially from 0 by factor of two
 - Narrow down when upper limit is discovered
- Optimization:
 - CAST(Number as varchar) and treat is as a string with numbers

bsqlshell.py

- I wanted to code something generic to forget about custom scripts
- I wanted to write optimum techniques as BSQL injection is **REALLY** slow
- Interactive shell + scriptable (Scapy like)
- bsqlshell.py is fast(er)!!
- Portable (python rocks!)

Demo

```
C:\Documents and Settings\Rodrigo Marcos\My Documents\python scripts\blindsql>
```



and more...

- Interaction sucks! I want something scriptable!

```
from bsishell import *  
pre = "http://www.vulnerable.com?id=1' and "  
post = " or '1'='2"  
user()  
table_enumeration()
```

Conclusions

- Blind SQL injection can be exploited and it really makes a difference
- The attack can be optimized for fewer requests to the database
- `bsqlshell.py` is quite cool (shameless propaganda)

Thanks

- Questions?