**SQL Injection**

SQL Injection의 종류로는 크게 In-Band SQL Injection과 Inferential SQL Injection로 구분할 수 있습니다.

- In Band SQL Injectioin

: 결과값을 얻을 수 있는 환경에서 진행되는 공격(classic SQL Injection)

eg) Union Based SQL Injection, Error Based SQL Injection

- Inferential SQL Injection

: 결과값을 얻을 수 없는 blind 환경에서 진행되는 공격

eg) Boolean Based SQL Injection, Time/Sleep Based SQL Injection

Union Based SQL Injection의 경우 앞의 결과와 뒤의 union으로 연결한 결과의 column의 수가 같지 않으면 오류가 발생합니다. 따라서 해당 column의 수를 미리 알 필요가 있는데 이때 하나씩 늘려가며 확인하는 방법도 있지만 order by를 사용할 수 있을 경우 해당 방법을 이용한 방법도 있습니다.

확인하는 방법의 경우 query 뒤에 order by [숫자]를 입력하였을 경우 오류가 발생하지 않으면 그 숫자 이상 혹은 이하에 해당하는 column이 사용된다는 것을 알 수 있습니다. 더 빠른 검색을 위해서는 binary search를 응용할 수 있을 것으로 보입니다.

MySQL의 경우 information\_schema DB의 schemata table에서 schema\_name을 조회하면 존재하는 DB의 이름들을 확인할 수 있습니다. schema\_name을 조건으로 하여 table\_name을 조회하면 해당 DB에 존재하는 table 이름들을 확인할 수 있습니다.

서로 다른 결과를 한 줄로 출력할 때는 group\_concat(항목) 함수를 통해서 한 줄로 표현할 수 있습니다.

Error Based SQL Injection의 경우 다양한 방법이 있는 것으로 보이지만 그 중 해당 강의에서 자주 사용하는 방법은 round(rand(0))과 having min(0)의 조합입니다. 오류가 발생하는 쿼리는 다음과 같습니다.

Group by round(rand(0)) having min(0)

이때 오류의 경우 Duplicate entry ‘1’ for key ‘group\_key’라고 출력되는데 ‘1’ 부분을 공격자가 원하는 내용을 출력되도록 변경할 수 있습니다. 만약 version 정보를 출력하고 싶은 경우 group by에서 concat(version(), round(rand(0)))과 같이 사용할 경우 ‘version 정보1’과 같이 출력됩니다.

BurpSuite를 사용할 경우 semi-auto SQL Injection이 가능합니다.

**SQL Injection LAB**

- LAB 1

첫 번째 환경의 경우 SQL Injection에 대한 아무런 방어도 없는 환경이었습니다. Keyword filtering도 없었기 때문에 ID에 “\’ or 1=1#” 을 입력함으로써 로그인 우회가 가능하였습니다.

기본적으로 로그인 우회가 가능한 것은 확인되었으나 이 외에도 여러 종류의 공격을 시도해보았습니다.

1. Error Based

Error Based SQL Injection을 시도해보면서 이 방법이 성공하기 위해서는 앞의 WHERE절이 TRUE가 만족될 필요가 있다는 것을 알 수 있었습니다. round(rand(0))과 having min(0)의 조합을 사용하였으며 payload의 경우 다음과 같이 작성하였습니다.

**“ 123' or 1=1 group by concat(version()," ",database(), " ", round(rand(0))) having min(0)# ”**

위 payload를 사용할 경우 아래와 같이 출력됨을 확인할 수 있었습니다.

**“ERROR during query execution: Duplicate entry '10.3.25-MariaDB-0ubuntu1 w 1' for key 'group\_key'1”**

database의 이름은 database() 함수를 사용하여 간단하게 알아낼 수 있지만 table name의 경우 별도의 함수가 존재하지 않습니다. 따라서 MySQL 기준 information\_schema database에 대해 subquery를 작성하여 이를 error 문구에 출력시키고자 하였는데 우선 다음과 같은 쿼리를 작성하였으나 결과를 얻는데 실패하였습니다.

**123' or 1=1 group by concat(version()," ",(select group\_concat(TABLE\_NAME) from information\_schema.TABLES), " ", round(rand(0))) having min(0)#**

- LAB 2

첫 번째 환경과의 차이점은 괄호()의 유무였습니다. 두 번째 페이지의 코드는 (id=’str’)과 같이 SQL Query가 작성되어 있었기 때문에 이를 우회하기 위해서는 다음과 같은 Query를 작성할 수 있었습니다.

**“') or 1=1#”**

- LAB 3

세 번째 환경은 첫 번째 환경을 기반으로 하되 html에서 username과 password 입력칸의 max length를 5로 제한하였습니다. 이 환경의 경우 브라우저의 개발자 도구를 사용하여 html code를 임의로 수정하여 기본적으로 지정된 값 이상의 쿼리를 작성할 수 있습니다. 혹은 burpsuite와 같은 proxy 도구를 사용하여도 쿼리를 임의적으로 조작하여 우회할 수 있습니다.

- LAB 4

네 번째 환경은 single quotes 하나를 입력하였을 때 이를 두 개의 single quotes로 str\_replace하여 single quotes의 입력을 필터링하는 환경이었습니다. 강의 영상에서는 \ 문자를 사용하여 일반 문자열로 인식하게끔 하여 str\_replace를 우회하였습니다.

**“\’ or 1=1-- -”**

위와 같이 query를 작성하였을 때, str\_replace를 거치며 전달되는 query문의 형태는 다음과 같이 변경됨을 알 수 있습니다.

**Password=’\’’ or 1=1-- -’**

SQL에서 \(backslash)의 경우 특수한 용도로 사용되는 특수문자들을 일반 문자로 인식되게끔하는 기능을 가지고 있습니다. 따라서 두 개의 single quotes 중 하나를 일반 문자로 인식하게끔 하여 이를 우회하는 것입니다.

- LAB 5

다섯 번째 환경은 str\_replace를 사용하되 single quotes를 없애는(“’”에서 “”로) 방법으로 사용되었습니다. 즉, ‘ or 1=1-- 와 같은 입력을 사용하였을 경우 결과적으로 or 1=1-- -와 같이 쿼리가 전송되는 것이었습니다. 이 문제의 경우 SQL의 구성을 이해할 필요가 있었습니다. SQL의 구조를 살펴보면 username과 password를 AND 조건으로 묶어 확인하는데 이때 username에 \(backslash)만 입력할 경우 아래와 같이 query문이 전송됩니다.

**Username=’\’ AND password=‘ ’**

LAB 4에서 확인할 수 있었듯이 backslash를 입력할 경우 특수문자를 일반 문자처럼 사용할 수 있게 되는데 이를 이용하여 username에 backslash만 입력할 경우 “‘AND password=” 가 문자열로 인식되게 됩니다. 이 점을 이용하여 기존 password를 입력하는 곳에서 **or 1=1-- -**와 같이 입력할 경우 Query문을 조작할 수 있게 됩니다.

- LAB 6

여섯 번째 환경은 Query의 결과값이 두 개 이상의 결과가 반환 되었을 경우 실패로 간주하여 공격을 필터링하는 것이었습니다. 이러한 필터링의 경우 자주 사용되는 방법인 “or 1=1-- -”에 대한 방어 기법으로 위 Query를 사용할 경우 table 내의 모든 결과를 반환하기 때문에 필터링에 걸리게 됩니다. 따라서 limit 구문을 추가하여 반환되는 결과를 1개로 제한해 이를 우회하였습니다.

**“’ or 1=1 limit 1-- -”**

위와 같은 필터링은 방어할 수 있는 Query가 제한적이기에 더 발전된 방법을 사용할 필요성이 있을 것 같다고 생각됩니다.

- LAB 7

일곱 번째 환경은 Query의 결과값을 count 함수를 통해 받고 그 반환값이 1일 경우에 참으로 간주하고 이 외의 경우는 로그인에 실패합니다. 현재 구성된 환경에서는 Database 구조를 직접 확인할 수 있었기 때문에 다음과 같이 공격 payload를 작성할 수 있었습니다.

**“’ or id=1-- -”**

위 Query는 사전 정보가 있다는 가정 하에 쉽게 작성할 수 있기에 union을 이용한 방식도 시도해 보았습니다. 결과적으로 다음과 같은 Query로 주어진 조건을 만족할 수 있었습니다. 또한 강의를 통해 limit 1,1 부분을 **“order by 1 desc”**로 대체하여 내림차순으로 정렬하여 조건을 만족할 수 있다는 것 또한 알 수 있었습니다.

**“' union (select 1) limit 1,1;-- -”**

강의 영상에서는 group by를 사용하여 다음과 같은 Query를 작성하였습니다.

**“’ or 1=1 group by username”**

- LAB 8

여덟 번째 환경은 preg\_match 함수를 사용하여 white space를 필터링하는 환경이었습니다. White space의 경우 filtering 할 수 있는 방법도 많고 그만큼 우회할 수 있는 방법 또한 다양합니다. 이 환경의 경우 가장 기본적인 괄호를 사용한 우회를 시도하였습니다. payload의 경우 다음과 같이 작성하였습니다.

**“'or(1=1)#”**

-- -를 사용하여 우회를 할 경우 공백이 필요하기 때문에 #을 이용하여 주석 처리를 하였습니다. 강의에서는 /\*\*/ 주석을 사용하여 white space 문자를 대체하였습니다.

- LAB 9

아홉 번째 환경은 php의 mysqli\_real\_escape\_string 함수를 사용한 필터링 환경이었습니다. 이 함수의 경우 특정 문자에 대하여 escape 문자인 \(backslash)를 추가하여 특수문자의 사용을 방지하기 위해 사용할 수 있는 함수입니다. 해당 함수에 대한 우회 방법을 찾아보았을 때 멀티 바이트 환경에서 사용 가능한 방법에 대해 찾을 수 있었으나 이 환경은 그렇지 않았기에 우회에 성공하지 못하였습니다. 따라서 강의를 참조하였는데 한 가지 간과하였던 점이 id 항목에 대해서는 single quotes로 묶여있지 않았기 때문에 id 부분에 다음과 같이 입력할 경우 쉽게 우회가 가능하였다는 점입니다.

**“1 or 1=1#”**

- LAB 10

열 번째 환경은 정규 표현식을 사용한 필터링 환경이었습니다. 사용된 정규 표현식은 다음과 같았습니다.

**^[0-9]+**

즉, 0에서 9에 해당하는 숫자로 시작하는 값이 아닐 경우 Query가 실패하는 것을 확인할 수 있었습니다. 또한 이 문제도 id 부분은 single quotes로 묶여 있지 않았기 때문에 아래와 같이 간단하게 우회할 수 있었습니다.

**0 or 1=1-- -**

- LAB 11

마지막 열한 번째 환경은 LAB 9와 LAB 10의 내용이 혼합되어 있었습니다. Password 입력의 경우 LAB 9에서 나왔던 mysqli\_real\_escape\_string을 통해 필터링 하였고 id의 경우 LAB 10을 응용한 시작 문자가 아닌 끝 문자를 검사하여 필터링 하였습니다. LAB 10에서 사용하였던 payload를 아래와 같이 변형하여 우회할 수 있었습니다.

**0 or 1=1-- -1**