오픈 API에서의 새로운 파라미터 요청 방식 제안

박재훈*, 서화정*†
*한성대학교 IT융합공학부 (대학원생)
*† 한성대학교 IT융합공학부 (교수)
p9595jh@gmail.com, hwajeong84@gmail.com

Suggestion of New Parameter Request Method for Open API

Jae-Hoon Park*, Hwa-Jeong Seo*†
*Division of IT convergence engineering, Hansung University (Graduate student)
*† Division of IT Convergence Engineering, Hansung University (Professor)

요 약

오픈 API에서는 사용자로부터 조회할 데이터를 요청을 통해 조건에 해당하는 데이터들을 선별하여 리턴하게 되는데, 현재 통용되는 방식은 다양한 조건을 설정하는 것에 있어 상당한 불편함이 따른다. 이에 따라 오픈 API에서 다양한 조건을 검색할 수 있는 방식을 제안한다. POST 메소드를 통해 숫자의 경우 원하는 검색 범위에 대한 설정을, 문자열의 경우 조건에 따라서 포함 혹은 일치하는 데이터를 검색한다. 이렇게 파라미터의 종류가 다양해짐에 따라 SQL 인젝션과 같은 보안에 대한 위험성도 커지며, 그것을 원천적으로 차단하기 위해 쿼리에 사용자로부터 받은 변수를 넣는 것이 아닌, 데이터베이스에서 얻은 데이터로부터 특정 알고리즘을 통해 사용자의 원하는 조건에 해당하는 데이터를 추출해내는 방법 또한 제안한다. 이를 통해 생산성 극대화를 기대한다.

1. 서론

오늘날 여러 개인 및 기관들에서 개발자들에게 필요한 정보를 제공해주기 위해 웹 서버를 이용하여 오픈 API를 제공하고 있다.[1] 개발자들은 이렇게 제공되는 정보들을 이용하여 새로운 서비스를 개발할 때에 이용할 수 있다. 오픈 API는 대부분 XML 혹은 JSON 형태의 데이터로 제공되며 보통 해당 API가 제공되는 서버에 GET 메소드를 통해 요청을 보내서 원하는 데이터를 제공받는 방식이다. 이 요청은 GET 메소드를 통하기 때문에 URI에 쿼리 형태로 파라미터를 담아서 보내는 형식이다.

기관에서 제공되는 오픈 API는 대부분 데이터를 제공할 때 인증키를 이용한다. 아무나 서버 자원에 접근하여 부하를 증가시키는 것을 막는 이유에서이다. 개발자는 서비스를 상용화하기에 앞서 오픈 API를 이용할 수 있도록 해당 기관으로부터 인증키를 얻어야 하며, 이렇게 얻은 인증키를 파라미터에 담아서 요청하는 것이다.

2. 파라미터 전송 방식

본 논문에서는 현재 사용되고 있는 파라미터 전송 방식에 대한 문제 제기 및 그에 대한 해결책을 제안한다.

2-1. 문제

오픈 API는 조회를 위해 활용되는 경우가 대부분이며, API를 제공하는 측에서 URI를 통해 요청하는 방식을 지정한다. 개발자는 이렇게 지정된 요청 방식을 통해서 URI 내에 쿼리 파라미터를 합친 뒤 해당 API에 요청하여 조건에 맞는 정보를 얻게 된다.

하지만 데이터의 형식과 종류가 다양함에 비해 URI를 통해 요청하는 방식은 표현이 다양하지 못하게 된다. GET 메소드를 통해 URI로 파라미터를 보낼 경우 ‘name1=value1&name2=value2&...’ 형태로 요청을 보내게 되는데, 이럴 경우 일치하는 값밖에 얻지 못하여 요청의 자유도가 떨어지며 API를 제공하는 측에서도 다양한 요청을 받아들이려면 또다른 방식을 제공해야 한다. 가령 어떤 오픈 API로부터 2020년 2월 29일에 해당하는 정보를 얻고 싶을 경우, 쿼리 파라미터(query parameter)의 경우에는 ‘uri.com?year=2020&month=2&day=29’ 형태로, 필수 파라미터(required parameter)의 경우에는

‘uri.com/2020/2/29’와 같은 형태로 요청을 하게 된다. 필수 파라미터의 경우에는 담을 수 있는 파라미터가 매우 제한적이므로, 보통 오픈 API에서는 쿼리 파라미터가 주로 사용된다. 하지만 이렇게 할 경우 원하는 형태의 조건 검색이 어렵다. 가령 ‘2월’이 아닌 ‘2월부터 5월까지’를 검색하고 싶을 경우 2, 3, 4, 5월 각각에 대해서 총 4개의 요청을 전송해야 한다. 서버측에서 따로 요구조건을 만들어놓지 않았을 경우 이렇게 요청이 낭비되는 문제가 발생한다.

이런 불편한 점들에 대해 좀 더 자유롭게 데이터를 얻을 수 있는 요청 방식을 제안한다.

2-2. 제안

파라미터를 헤더에 담아서 전송하는 GET 메소드보다 바디에 담아서 전송하는 POST 메소드를 이용할 것을 제안한다. 그리고 요청의 바디 내부에 JSON 형태의 데이터를 담는다. JSON 데이터에는 컬럼의 타입에 맞춰서 문자열, 숫자, 불리언 등에 대한 각각의 형식에 따라서 데이터를 전송한다.

표 1. 숫자 검색에 대한 형식

변수명	타입	상세
over	number	초과
above	number	이상
below	number	이하
under	number	미만
same	number	동일

표 2. 문자열 검색에 대한 형식

변수명	타입	상세
allCaseAllow	boolean	대소문자 구분여부
str	string	검색할 문자열

```
{
    "birth_year" : {
        "above" : 1995,
        "under" : 2001
    },
    "email" : {
        "allCaseAllow" : false,
        "str" : "%example%"
    }
}
```

Fig 1. 요청 JSON 데이터 예시

문자열 검색에서 str은 SQL의 LIKE 구문과 같이 ‘%’를 통해 포함 여부를 결정한다. 가령 str을 ‘hello’라고 설정하면 ‘hello’에 해당하는 데이터만 리턴되지만, ’%hello%’라고 설정하면 ‘hello world’ 등의 ‘hello’가 포함된 문자열이 리턴되게 된다. 원하는 위치에서 검색하기 위해 ‘%’를 앞 혹은 뒤에만 설정할 수도 있다. 이를 통해 좀 더 다양한 검색이 가능하게 한다.

Fig 1에서의 JSON 데이터 예시는 태어난 연도가 1995 이상, 2001 미만이며 이메일에 ‘example’이 포함된 사람들 검색하는 예시이다. allCaseAllow 옵션이 false로 설정되어 있기에 ‘eXample’ 등의 대소문자가 다른 경우는 검색 결과에서 배제된다.

숫자 검색 조건에서 ‘above’와 ‘over’을 동시에 사용할 경우 두 조건의 교집합이 되는 ‘above’에 해당하는 결과가 검색된다.

3. 보안

오픈 API에서는 파라미터를 통해 받은 문자열을 충분한 검증을 거치지 않고 데이터베이스에 대입할 경우 보안에 관련한 문제를 야기할 수 있다. 이에 대한 문제점 및 해결책을 제안한다.

3-1. 문제

관계형 데이터베이스를 이용할 경우 대부분 쿼리문을 조합하여 데이터베이스에 실행시킨 뒤 결과값을 받아서 이용하는 방식을 가진다. 이는 올바르지 않은 문자가 포함될 경우 SQL 인젝션이 생길 가능성을 가지게 된다.

가령, 회원 로그인을 위해 사용자가 아이디와 비밀번호를 form을 통해 전송할 경우 서버에서는 그것을 검증하기 위해 SELECT * FROM users WHERE ID = ‘전송된 ID’ AND PASSWORD = ‘전송된 비밀번호’ 쿼리를 실행시켜서 결과가 있을 경우 얻어진 결과값을 이용하여 로그인을 진행시키고, 결과가 없을 경우에는 아이디 혹은 비밀번호가 틀렸다는 경고를 띄워주게 된다. 하지만 만약 사용자가 아이디를 ADMIN, 비밀번호를 1234’ AND ‘1’=’1 이라고 전송할 경우 쿼리의 형태는 SELECT * FROM users WHERE ID = ‘ADMIN’ AND PASSWORD = ‘1234’ AND ‘1’=’1’이 되어, ‘1’=‘1’에 의해 무조건 TRUE가 되면서 ADMIN 계정으로 로그인 할 수 있게 된다. 이러한 SQL 인젝션에 대해서 프로그래머가 확실한 대비책을 구현해야 한다. 싱글 쿼테이션(‘)이 포함되어 있을 경우 실행시키지

않는 방법도 있지만, 어디까지나 그러한 처리를 직접 해줘야 하는 것이고, 이는 프로그래머의 실수로 처리가 되어 있지 않을 경우 문제가 생길 수 있다.

ORM(Object Relational Mapping)을 이용할 경우에도 인젝션을 야기할 수 있다. ORM은 데이터베이스 테이블을 객체 형태로 다루게 되어서 통상적인 사용의 경우에는 인젝션이 일어날 일이 없게 되지만, ORM을 구현하는 여러 라이브러리들은 복잡한 쿼리 작업은 객체 형태가 아닌 직접적인 SQL을 이용하여 구현할 수 있도록 방안을 마련해놓고 있다. 대표적으로 Hibarnate의 HQL이 그러한 것인데, 이것을 이용할 때에 인젝션이 일어날 수 있다.

SQL을 아예 이용하지 않는 NoSQL에서도 인젝션 문제를 원천적으로 피할 수는 없게 된다. 대표적인 NoSQL인 MongoDB의 경우 검색을 할 때에 문자열을 넣어서 하는 방식을 이용하는데, 이때 사용자로부터 받은 문자열을 직접 넣을 경우 인젝션이 일어날 수 있다. 가령 아이디를 보냈을 경우 {id: '사용자로부터 받은 id값'} 형태의 쿼리를 수행한다고 가정했을 때, 해커가 MongoDB의 명령어인 $ne나 $exists와 같은 구문을 문자열에 추가시킬 경우 원하는 정보를 빼갈 수 있게 된다.

이렇듯 다양한 데이터베이스에서 해커는 인젝션을 일으켜 서버에서 강제로 정보를 빼갈 수가 있는데, 이러한 인젝션의 예방이 충분하지 않게 되면 뚫릴 수밖에 없게 된다. 특히나 오픈 API와 같이 여러 변수를 사용자로부터 직접 입력받는 서비스의 경우 더욱 보안에 취약해지게 된다. 그렇기에 이러한 공격으로부터 원천적으로 방어할 수 있는 수단이 필요하다.

3-2. 제안

쿼리문에 사용자로부터 받은 변수를 직접 넣지 않는 방법을 제안한다. 데이터베이스에서 쿼리문을 실행시키려 할 때 사용자로부터 받은 변수를 직접 담은 쿼리문을 이용할 경우 전부 인젝션 방어용 처리를 해주지 않는 경우 결국 인젝션 문제가 일어날 수밖에 없게 된다. 이러한 문제를 원천적으로 차단하기 위해서 데이터베이스로부터 데이터를 받을 때부터 조건을 거는 것이 아닌, 별다른 조건을 걸지 않은 채로 데이터를 받은 뒤 그것을 조건에 따라 걸러내어 사용자에게 리턴시키는 방식을 사용한다.

예를 들어 로그인 할 때에 사용자로부터 아이디와 비밀번호를 전송받을 경우, 쿼리문 내부에 ID = '입력받은 ID' AND PASSWORD = '입력받은 비밀번호'와 같은 조건을 거는 것이 아닌, SELECT * FROM USERS와 같이 조건이 걸려있지 않는 쿼리문을 실행시킨 뒤 받은 사용자 데이터로부터 반복문과 조건문을 이용하여 사용자를 확인하고 로그인을 수행하는 것이다.

이러한 방식을 상기한 오픈 API에서의 파라미터를 구체화하는 방식에 적용시킴으로써 오픈 API에서 생길 수 있는 인젝션 문제를 예방할 수 있다. 서버에서 받을 수 있는 형식은 정해져 있으므로 코드를 통해 데이터를 분류하는 작업도 마찬가지로 정형화 될 수 있다.

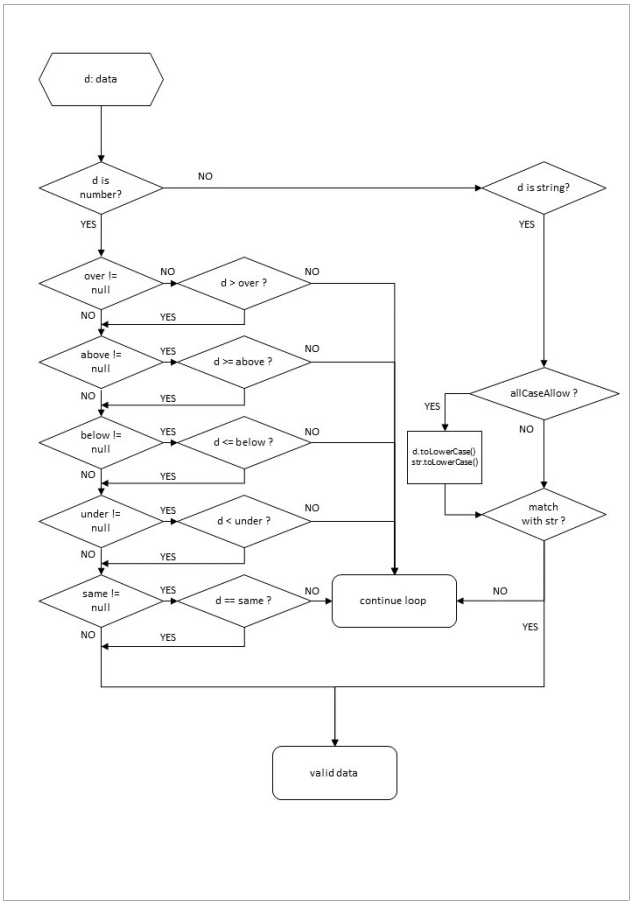

Fig 2. 조건 확인 알고리즘 순서도

데이터베이스를 거쳐서 나온 결과 배열값으로부터 루프를 돌면서 Fig 2의 알고리즘을 거쳐서 조건에 맞는 데이터들만을 골라내어 사용자에게 리턴시켜준다. 이렇게 될 경우 인젝션 공격으로부터 원천 차단할 수 있게 된다. 다만 해당 데이터의 타입이 무엇인지에 대한 확실한 명시가 필요하다.

다량의 데이터를 처리하는 것에 따라 서버의 CPU 사용량이 증가하는 것이 우려된다면, 검색 조

건에 페이징을 추가하여 각 요청에 대해 처리하는 데이터의 개수를 제한하는 방식을 이용할 수 있다. 가령 한 페이지에 30개의 데이터를 갖고 있다고 가정한다면, 페이징 파라미터에 따라 몇 번째 페이지인지를 알아내고 그에 해당하는 데이터들을 데이터베이스에서 얻어낸 뒤 30개의 데이터에서 조건 확인 알고리즘을 거친 뒤 사용자에게 결과를 리턴해주면 된다.

4. 결론

데이터를 다루는 서비스가 증가함에 따라 다양한 오픈 API의 사용도 증가하고, 그에 따라 복잡한 조건 검색 및 향상된 보안을 요구하게 되었다. 본 논문에서는 오픈 API에서의 타입에 따른 다양한 검색 조건 설정과, 그러한 방식을 적용하면서 SQL 인젝션 공격에 더 취약해질 수 있음에 따라 인젝션 공격으로부터 방어할 수 있는 방식을 제안하였다.

최근 RESTful API[2] 사용이 증가함에 따라 RESTful API의 조회에서도 이러한 방식을 이용하는 것을 기대한다. 조회 화면이 많은 어플리케이션의 경우 이러한 방식을 통해 생산성을 극대화 할 수 있을 것이다.

참고문헌

[1] 공공 데이터 이용 가이드 [Internet], https://www.data.go.kr/guide/guide/guide.do.

[2] R. T. Fielding, “Architectural styles and the design of network-based software architectures,” Ph.D. dissertation, Information and Computer Science, Univ. of California, IRVINE, 2000.