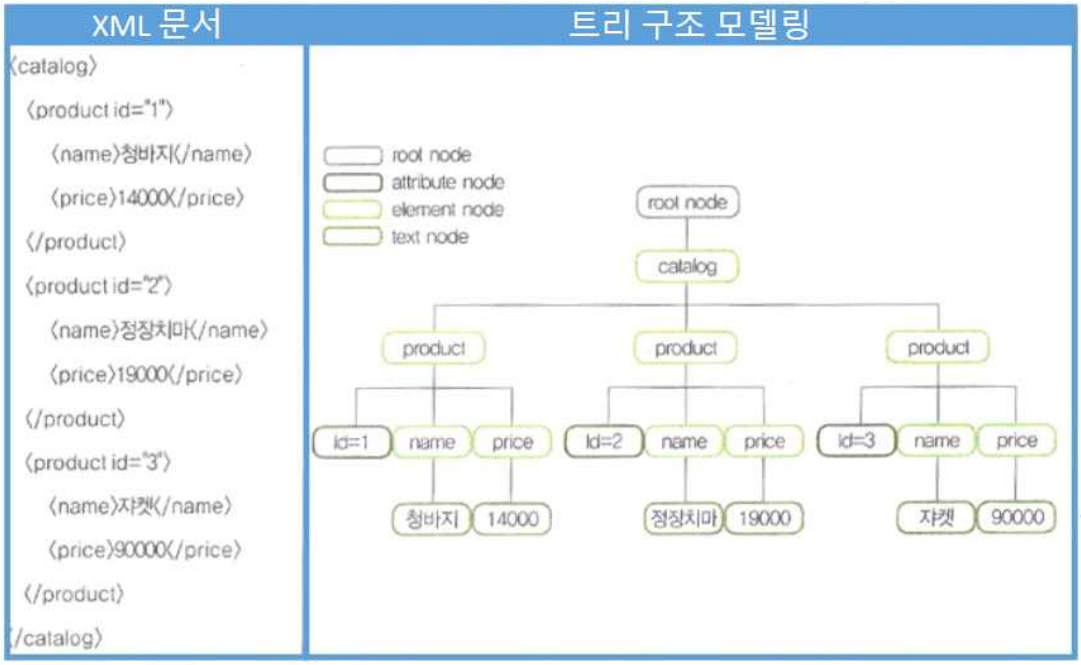
과목명	시큐어코딩
담당교수	우사무엘 교수님
학과	소프트웨어학과
학번	32153180
이름	이상민
제출일자	2020.06.09

1. 입력값 검증 부재로 인한 삽입 취약점 실습

- XPath : XML 문서에서 특정 요소나 속성까지 도달하기 위한 경로를 요소의 계층을 이용해 표현하는 것



- XPath 삽입 : XML 문서에 저장된 데이터를 애플리케이션에서 검색하거나 읽기 위해 사용하는 표현 방식

- 취약점 발생 원인 : XPath 쿼리문을 생성할 때, 입력값에 대한 검증 작업을 수행하지 않고 동적으로 생성되는 XPath문에 삽입해 사용하는 경우 취약점 발생

(1) XPath 삽입 공격

입력값이 XML 쿼리의 조회 키로 사용되는지 체크

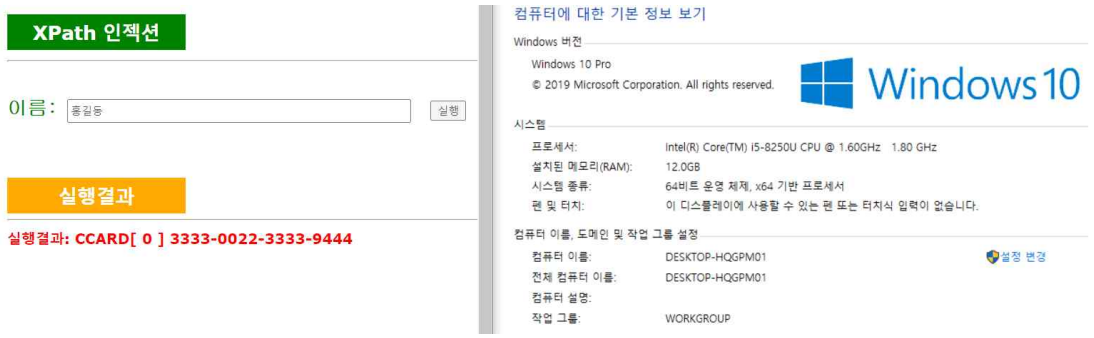
- 입력값 ‘ or ’‘=’ 을 사용하여 모든 사용자의 CCARD 정보 유출

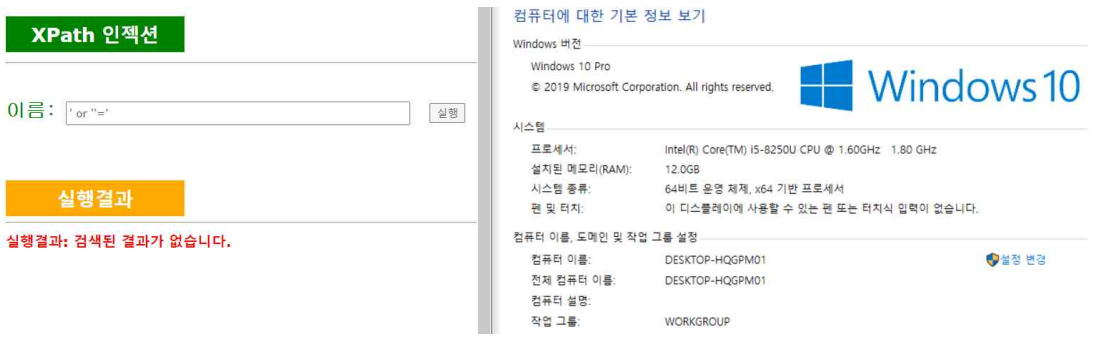
(2) XPath 삽입 방어

XPath에서 사용되는 외부 입력값에 대해 안전한 값으로 필터링해 사용

```
public String XPathFilter(String input) {
    return input.replaceAll("[ ',\\[]", "");
}
```

```
System.out.println("ccard 출력");
// String expression = "/addresses/address[@name='"+name+"']/ccard";
String expression = "/addresses/address[@name='"+XPathFilter(name)+"']/ccard";
```





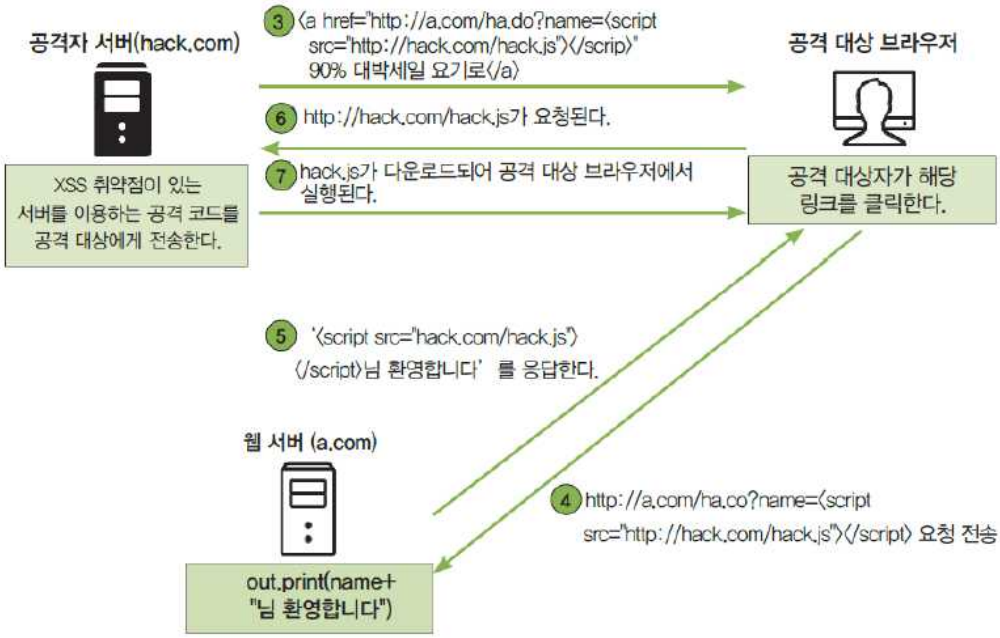
- XPathFilter() : XPath 삽입을 발생시킬 수 있는 문자 필터링
  쿼리문의 의미를 바꿀 수 있는 특수 문자를 공백으로 변경

따라서 ‘ or ’‘=’ 구문을 이용하여 허가되지 않은 파일의 정보를 조회하려 할 때, XPathFilter() 함수에 의해 입력값이 필터링된다. 결과적으로 안전한 값만 프로그램에서 사용하도록 한다.

2. 크로스 사이트 스크립팅 취약점 실습

- 크로스 사이트 스크립팅(XSS) 취약점 : 외부 입력값이 충분한 검증 없이 동적으로 생성되는 응답 페이지에 사용되는 경우 발생

- Reflective XSS : 공격자가 악성 스크립트를 입력값으로 전달하도록 만들어진 URL을 사용자가 클릭하도록 유도



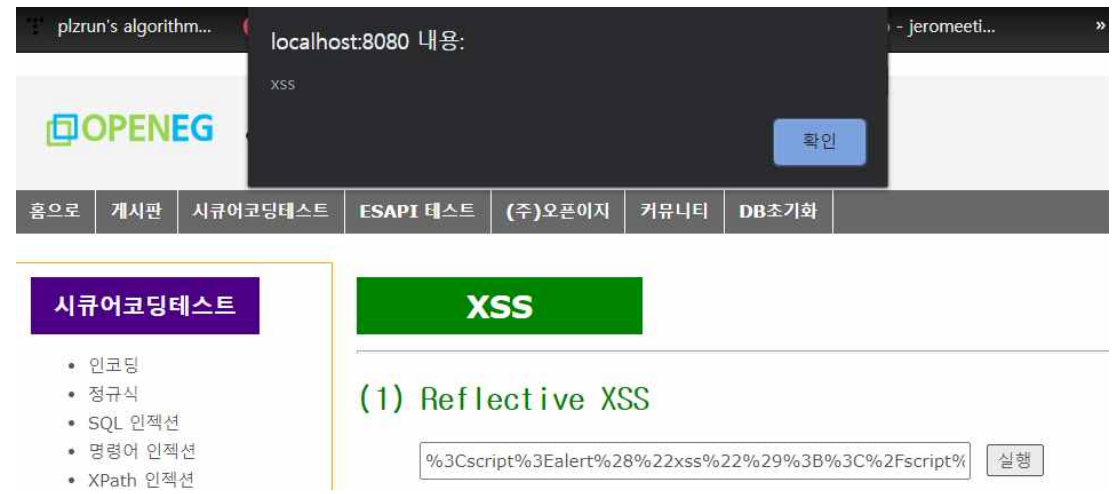
- Stored XSS : 악성 스크립트를 DB에 저장해 시스템을 사용하는 모든 사용자들이 해당 스크립트를 실행하게 함 -> 사용자 쿠키 정보 탈취 및 악성 사이트로 이동

- DOM XSS : AJAX 프로그램에서 사용되는 자바스크립트를 이용해 브라우저에 수신된 데이터를 다시 잘라 write 작업을 수행하는 경우 XSS 공격이 가능하게 함

- 취약점 발생 원인 : 외부 사용자의 입력값이나 DB에서 검색한 결과값을 검증하지 않고 사용한 경우

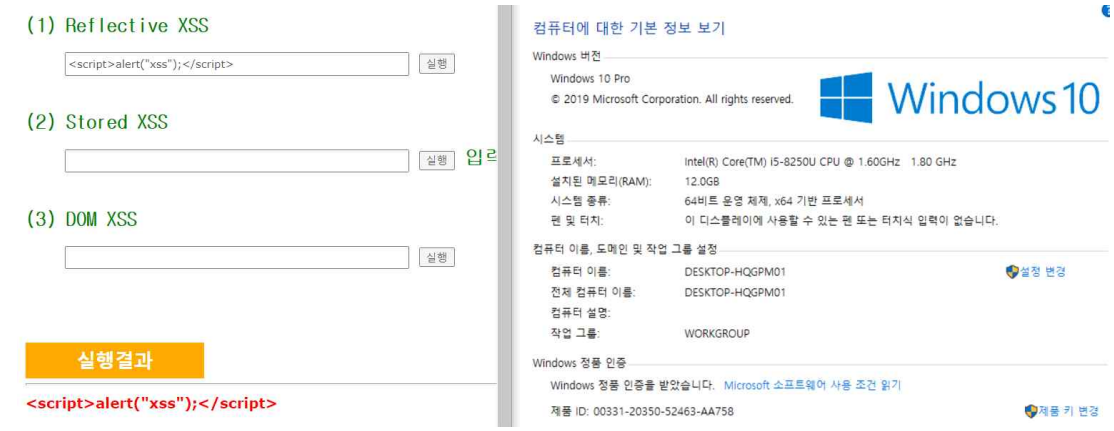
(1) Reflective XSS 공격

입력값을 출력에 사용하는 기능에서 Reflective XSS 발생 가능성 체크



(2) Reflective XSS 방어

오픈소스 라이브러리 활용하여 입력값에 대해 XSSFilter 적용



```
// Reflective XSS 테스트
@RequestMapping(value="/test/xss_test.do", method = RequestMethod.POST)
@ResponseBody
public String testXss(HttpServletRequest request) {
   StringBuffer buffer=new StringBuffer();
   String data=request.getParameter("data");

   try {
      data=URLDecoder.decode(data, "UTF-8");
      System.out.println("Data: "+data);
   } catch (UnsupportedEncodingException e) {
      System.out.println(e);
   }
   XssFilter filter=XssFilter.getInstance("lucy-xss-superset.xml");
   buffer.append(filter.doFilter(data));
   return buffer.toString();
}
```

- 각종 방식으로 인코딩되어 전달되는 데이터를 필터링하기 위해서는 인코딩 규칙까지도 감안하여 만들어야 함 -> XSS Filter를 만드는 것보다 이미 잘 검증된 필터 활용 권장

- data=URLDecoder.decode(data, “UTF-8”);
인코딩되어 입력되는 값은 디코딩 후 필터 적용

- XSSFilter filter=XssFilter.getInstance(“lucy-xss-superset.틱”);
필터 객체 생성 후 출력값에 대해 Anti-XSS 필터 적용

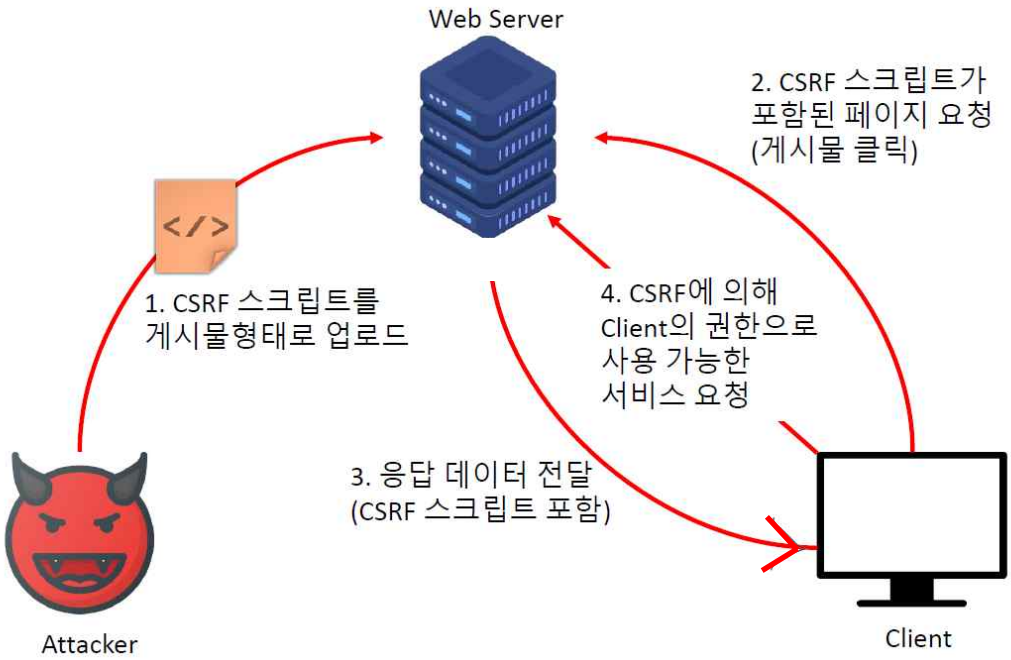
따라서 사용자가 요청을 한 입력값에 대해 XSS Filter를 적용하여 안전한 값만 전달한다. 그 후 interceptor가 입력값을 검증하여 안전하지 않은 입력값에 대해서는 요청을 차단하고, 안전한 입력값은 넘긴다.

3. 크로스 사이트 요청 위조 취약점 실습

- 크로스 사이트 요청 위조(CSRF) : 서버가 클라이언트의 요청이 인증받은 사용자의 인가된 실제 요청인지 구분하지 않고 요청을 처리하는 경우 발생

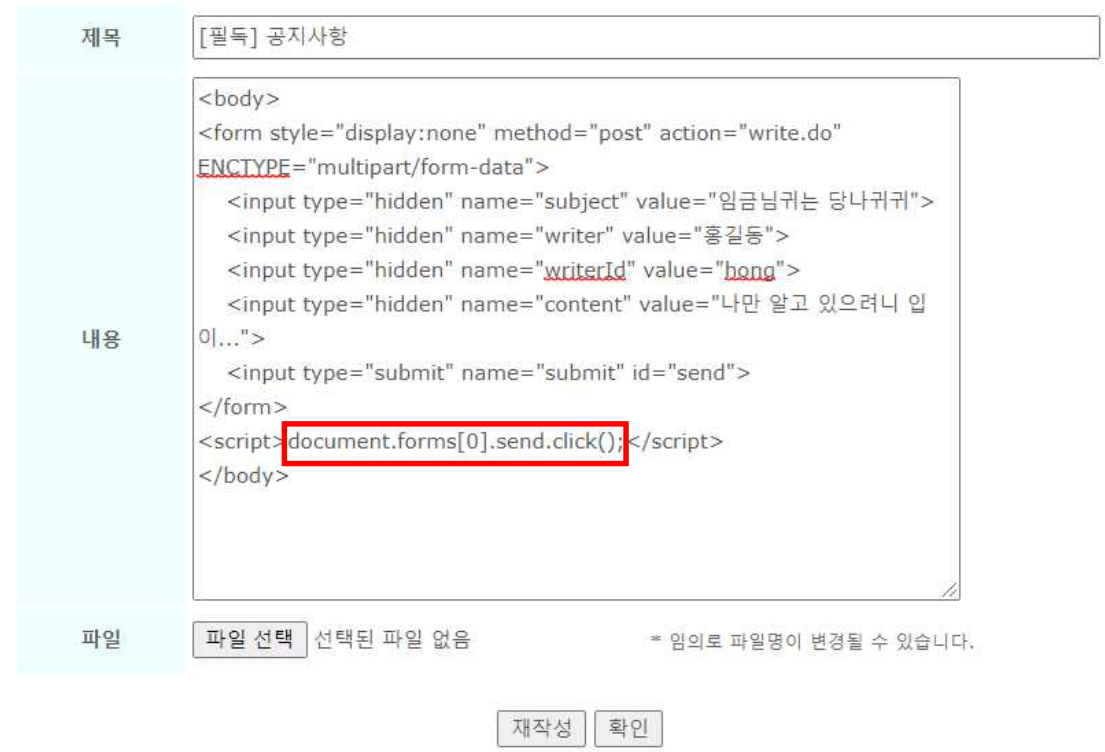
- 공격자가 작성해 놓은 request를 통해 일어나는 악의적인 공격으로 자신의 의도와는 다른 현상 발생

- 취약점 발생 원인 : 많은 웹사이트에서 사용자가 보내는 요청 데이터가 정상적인 경로를 통한 요청인지의 여부는 구분하지 않는 경우가 많음



(1) 크로스 사이트 요청 위조 공격

조작된 Form을 이용하여 전송된 요청이 정상처리 되는지 체크



글번호	제목	작성자	댓글수	조회수	추천수	작성일
1	[필독] 공지사항	테스트	0	0	0	2020-06-09 00:00:00

글번호	제목	작성자	댓글수	조회수	추천수	작성일
1	임금님귀는 당나귀귀	홍길동	0	0	0	2020-06-09 00:00:00
2	[필독] 공지사항	테스트	0	1	0	2020-06-09 00:00:00

- 게시판 목록에서 CSRF 공격 코드를 담고 있는 공지사항 글을 클릭하여 자동 글쓰기가 실행되는 것을 확인

- document.forms[0].send.click()이 실행될 때 폼 데이터는 write.do로 요청 전달

(2) 크로스 사이트 요청 위조 방어

- CSRF 취약점을 제거하기 위해 기본적으로 화면을 요청할 때는 GET 방식으로, 데이터가 전달되어서 처리되어야 하는 요청은 POST 방식으로 요청이 전달되도록 설계

- 데이터 처리를 위한 요청

① 사용자 -> ② 작업메뉴선택 -> ③ 입력페이지응답 -> ④ 데이터 입력 -> ⑤ 처리요청 -> ⑥ 처리 -> ⑦ 결과반환

CSRF 취약점은 미리 작성된 스크립트를 이용하여 '⑤ 처리요청'을 실행하였을 때 '⑥ 처리'에서 ①~④ 작업의 생략을 탐지하지 못한 경우 발생

고로 이 취약점을 제거하기 위해 '⑥ 처리'에서 ①~④ 절차를 거쳐 전달된 요청인지를 확인하고 처리하도록 구현하면 된다.

실제 사용자가 전송한 요청인지 확인하는 절차 추가

```
<mvc:interceptors>
    <mvc:interceptor>
        <mvc:mapping path="/board/write.do"/>
        <bean class="kr.co.openeg.lab.common.interceptor.CSRFInterceptor"/>
    </mvc:interceptor>
</mvc:interceptors>
```

- 사용자의 작업 요청을 입력받기 위한 화면을 응답할 때 CSRF 토큰을 히든값으로 보낼 수 있도록 삽입

따라서 CSRF 토큰값이 포함된 요청이 수신되면 이 값을 먼저 검사하여 요청의 유효성을 체크한 뒤 유효한 요청인 경우에만 게시글이 저장되도록 한다.