CSRF 공격

202112021 박채우

쿠키

쿠키란 이용자의 신원 정보가 포함되어 있는 일종의 서명과 같은 역할을 한다.

즉 웹 사이트에서 이용자의 정보가 포함되어있는 쿠키란 클라이언트에서 보내진 요청이 이용자로부터 왔으며 해당 요청에 대해 이용자가 동의했고 서버는 해당 요청에 이용자의 권한으로 요청을 수행하게 된다.

CSRF

CSRF(Cross-Site Request Forgery) 는 사이트 간 요청 위조 공격을 의미한다. 사용자가 자신의 의지와는 무관하게 공격자가 의도한 행위(수정, 삭제, 등록 등)을 특정 웹사이트에 요청하게 하는 공격을 의미한다.

인증된 사용자가 웹 애플리케이션에 특정 요청을 보내도록 유도하는 공격 행위를 일컫는다.

취약점 발생

해당 공격의 취약점은 특정 웹 사이트가 사용자의 웹 브라우저를 신용하는 상태일 때의 취약점을 이용한다.

즉 웹 사이트가 사용자를 신뢰하는 상태일 때 사용자의 입력값 검증 및 사용자의 요청 시 세션 쿠키, 출발지 주소 등을 올바르게 요청 된 것인지 검증을 하지 않아 발생한다.

=> 특정 사용자가 웹 사이트에 로그인 한 상태에서 해당 공격을 시작한다면 웹사이트는 해당 공격 명령이 믿을 수 있는 사용자로부터 요청 된 것이라고 판단한다.

CSRF

크로스 사이트 요청 위조 공격은 사용자의 정보 탈취보다는 특정 작업을 무단으로 진행하기 위한 목적으로 이루어지는 경우가 많다. 보통 최종 사용자 데이터의 수정, 삭제 등을 수행하지만 만약 최종 사용자가 관리자 계정인 경우 CSRF 공격으로 전체 웹 프로그램 자체를 손상시킬 수 있다.

CSRF

CSRF 공격의 전제 조건

1. 사용자(피해자)가 보안이 취약한 서버로부터 이미 인증을 받은 상태여야한다.
2. 쿠키를 기반으로 서버의 세션 정보를 획득할 수 있어야 한다. //수정하기
3. 공격자는 서버를 공격하기 위해 해당 요청 과정을 이해해야 한다.

XSS 공격과의 비교

1. XSS 공격과 CSRF 공격은 사용자의 브라우저를 이용해서 공격을 수행한다는 공통점이 있다.
2. XSS 공격과 CSRF 공격 둘 다 악성 스크립트를 삽입해서 공격을 수행한다.

XSS 공격과의 비교

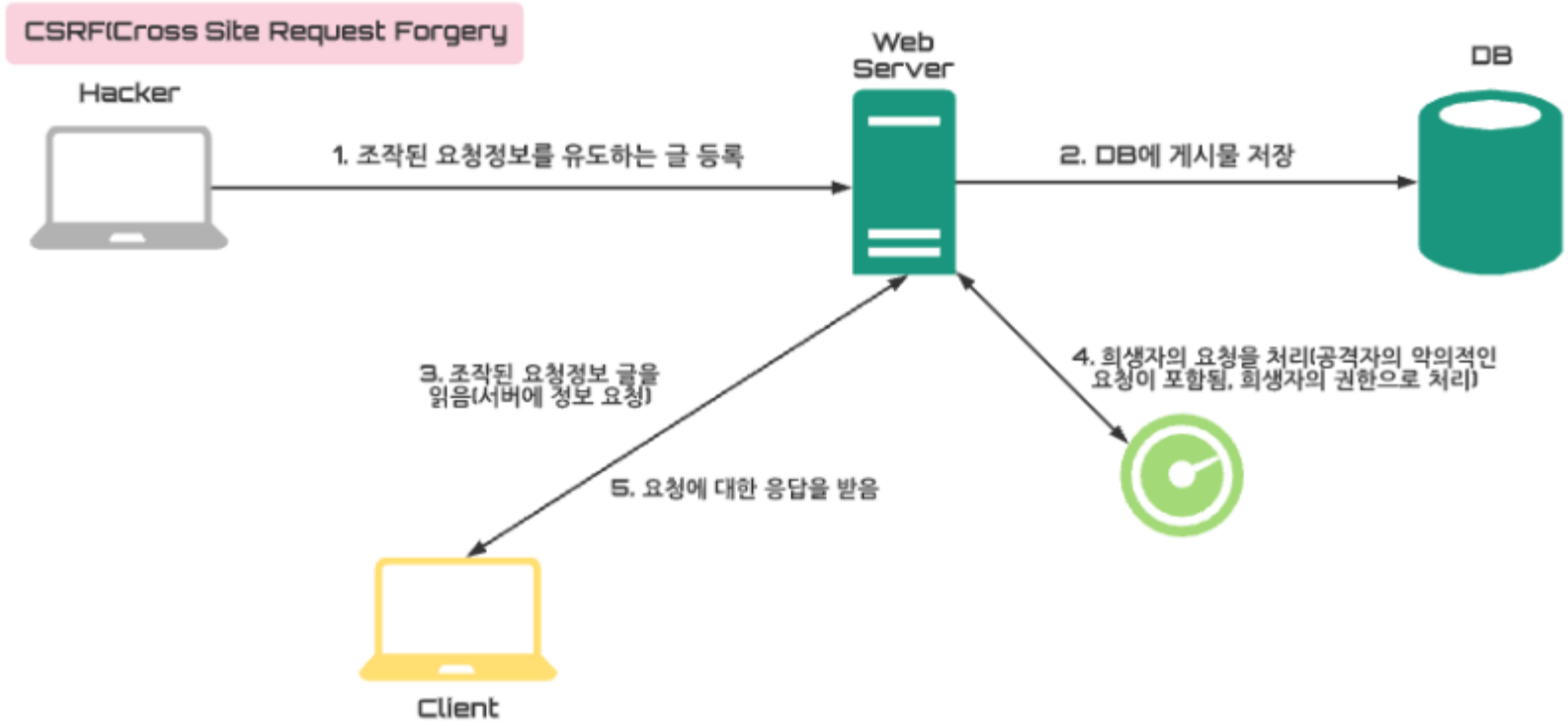
1. CSRF 공격은 사용자의 인증된 세션을 악용하는 공격 방식인 반면 XSS 공격은 인증된 세션 없이도 공격을 진행할 수 있다.

2. CSRF 공격은 사용자의 요청을 위조함으로써 사용자 몰래 송금, 게시글 작성 등 특정 행위를 수행하는 것을 목적으로 하지만 XSS 공격은 사용자의 PC에서 스크립트를 실행해 사용자의 정보를 탈취한다.

XSS 공격과의 비교

	XSS	CSRF
공격대상	Client	Server
목적	스크립트 실행이 목적	특정 행동을 유도
공격방법	인증된 세션 없이 공격 ex)악성코드 사이트로 리다이렉트, 인증(로그인)없이 피해를 입힐 수 있음	인증된 세션을 악용하여 공격 ex)반드시 인증된 사용자를 공격

공격 원리


CSRF(Cross Site Request Forgery
Hacker
Web Server
DB
1. 조작된 요청정보를 유도하는 글 등록
2. DB에 게시물 저장
3. 조작된 요청정보 글을 읽음(서버에 정보 요청)
4. 희생자의 요청을 처리(공격자의 악의적인 요청이 포함됨, 희생자의 권한으로 처리)
5. 요청에 대한 응답을 받음
Client

공격 시나리오

1. 공격자는 게시판에 악의적인 코드를 포함한 게시물을 등록한다.
2. 사용자가 악성코드가 삽입된 게시물을 확인한다.
3. 사용자의 권한으로 해당 악성코드의 요청이 수행되고 서버는 정상적인 요청이라 판단 후 요청을 수행한다.
4. 공격자가 수행하고자 하는 공격이 수행된다.

공격 시나리오

https://any-bank-site.com/sendMoney?to=hacker&amount=100000

피해자가 해당 글을 열람했을 때 이미 인증이 되어있는 상태이므로
서버는 해당 요청을 수락하게 된다.

실습

Title: test
Message: test
Submit

```
POST /WebGoat/attack?Screen=128&menu=900 HTTP/1.1
Host: localhost:8080
Content-Length: 37
Cache-Control: max-age=0
Authorization: Basic Z3Vlc3Q6Z3Vlc3Q=
sec-ch-ua: "Chromium";v="105", "Not)A;Brand";v="8"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
Origin: http://localhost:8080
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/105.0.5195.102 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Referer: http://localhost:8080/WebGoat/attack?Screen=128&menu=900
Accept-Encoding: gzip, deflate
Accept-Language: ko-KR,ko;q=0.9,en-US;q=0.8,en;q=0.7
Cookie: JSESSIONID=3B545A563E84254C5EE07A3B1670CE05
Connection: close

title=test&message=test&SUBMIT=Submit
```

실습

Look Me

```
<img src="http://localhost:8080/WebGoat/attack?
Screen=128&menu=900&title=attack&message=attack&SUBMIT=Submit"
width="1" height="1" />
```

Message Contents For: attack
Title: attack
Message: attack
Posted By:guest

Message List
test
Look Me
attack

실습

초기 상태 ID : admin PW:password

```
GET /dvwa/vulnerabilities/csrf/?password_new=normal&password_conf=normal&Change=Change HTTP/1.1
Host: localhost
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101 Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://localhost/dvwa/vulnerabilities/csrf/
Cookie: security=low; PHPSESSID=4s2j54gf5f0kfc9njejo4o23v4
Connection: close
Upgrade-Insecure-Requests: 1
```


```
<img
src=http://127.0.0.1/dvwa/vulnerabilities/csrf/?password_new=Hackin
g&password_conf=Hacking&Change=Change>
```

예방대책

1. 토큰 생성하기 : JSP 에서 사용자의 세션이 임의의 난수 값을 저장하고 해당 사용자가 어떠한 요청으 보낼 때 마다 난수 값을 포함시켜 서버에 전송한다. 이후 서버는 요청을 받을 때 마다 세션에 저장된 토큰값과 요청 파라미터에 전달되는 토큰 값이 일치하는지 검증한다.

예방대책

클라이언트

```
session.setAttribute("CSRF_TOKEN",UUID.randomUUID().toString());


<input type="hidden" name="_csrf" value="${CSRF_TOKEN}" />
```

서버

```
String param = request.getParameter("_csrf");


if (request.getSession().getAttribute("CSRF_TOKEN").equals(param)) {
            return true;
}

else {
            response.sendRedirect("/");
            return false;
}
```

예방대책

2. 레퍼러 확인하기 : 서버 단에서 request의 레퍼러를 확인하여 도메인이 일치하는지 검증하는 방식이다. 즉 같은 도메인 상에서 들어오는 접속 또는 요청은 허용하나 갑자기 예상치 않은 곳에서 요청을 호출 할 때 해당 요청을 차단하는 것을 의미한다.

예방대책

```
String referer = request.getHeader("REFERER");
if( referer != null && referer.length() > 0){
  out.println("referer : " + referer);
  out.println("You can enter");
}else{
  out.println("You can not enter");
}
```

```
var hostName = document.referrer;
if(hostName =! 'https://.com'){
    alert('Hello admin!');
}else{
    alert('Who are you?');
}
```

예방대책

3. Samesite 설정하기 : Samesite 를 Lax 또는 Strict 로 지정해주면 Cross-orgin 환경(도메인 주소는 같을 지라도 포트번호가 다른 경우)에서는 쿠키의 전송이 불가능해진다. 즉 이를 이용해서 CSRF 공격을 예방 할 수 있다.

```
Set-Cookie: auth=          ; SameSite=Lax;
```

4. Host Prefix : 토큰값을 단순히 쿠키로만 설정한다면 해당 전송은 공격에 상대적으로 취약해진다. 이 때 토큰에 Host prefix을 사용하는 경우 구형 IE를 제외한 대부분의 브라우저에선 해당 Prefix를 도메인을 식별하는 장치로 쓰일 수 있다.

```
Set-Cookie: __Secure-ID=123; Secure; Domain=example.com
Set-Cookie: __Host-ID=123; Secure; Path=/
```