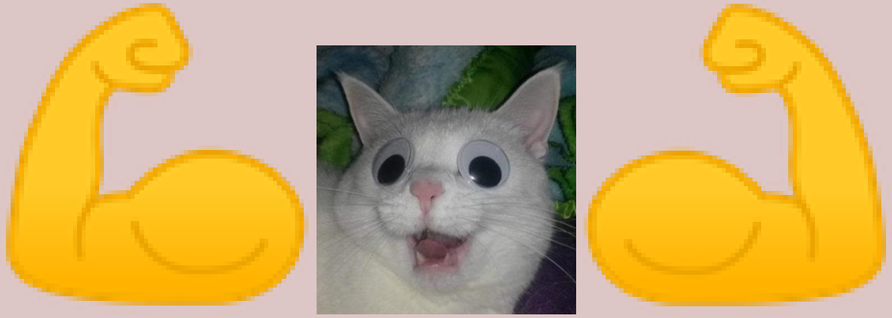
가볍게 웹 취약점을 알아보자

with 실제 시연 (XSS/SQL Injection/Path Traversal)



> 발표 자료나, 시연 서버 코드는 Slack → #92 자율-부캠-라디오를 참고해주세요 !

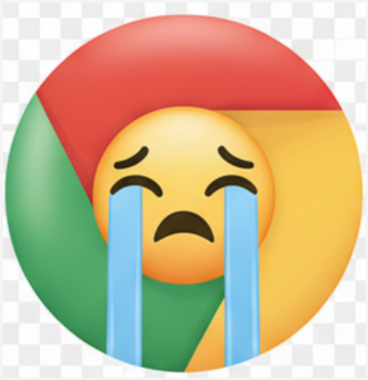
웹 풀스택 J123_박준호

이런 큰 기회를 주신 J277_최현호 님에게 무한한 감사를 올립니다...

목차

XSS(Cross Site Scripting)

XSS(Cross Site Scripting)
웹 상에서 기초적인 취약점 공격 방법의 일종입니다.
악의적인 사용자가 공격하려는 사이트에 스크립트를 넣는 기법을 말합니다.



외부에서 들어온 데이터를 적절히 처리하지 못해서,
사용자 브라우저에서 스크립트로 실행되는 취약점입니다.

만약 XSS가 성공한다면, 공격자는 사용자의 쿠키나 로컬 스토리지를 탈취한다던가,
화면을 조작해서 피싱 페이지로 리다이렉트 한다던가, 임의의 요청을 보낸다던가,
스크립트를 활용해 다양한 행동을 수행하게끔 할 수 있습니다.

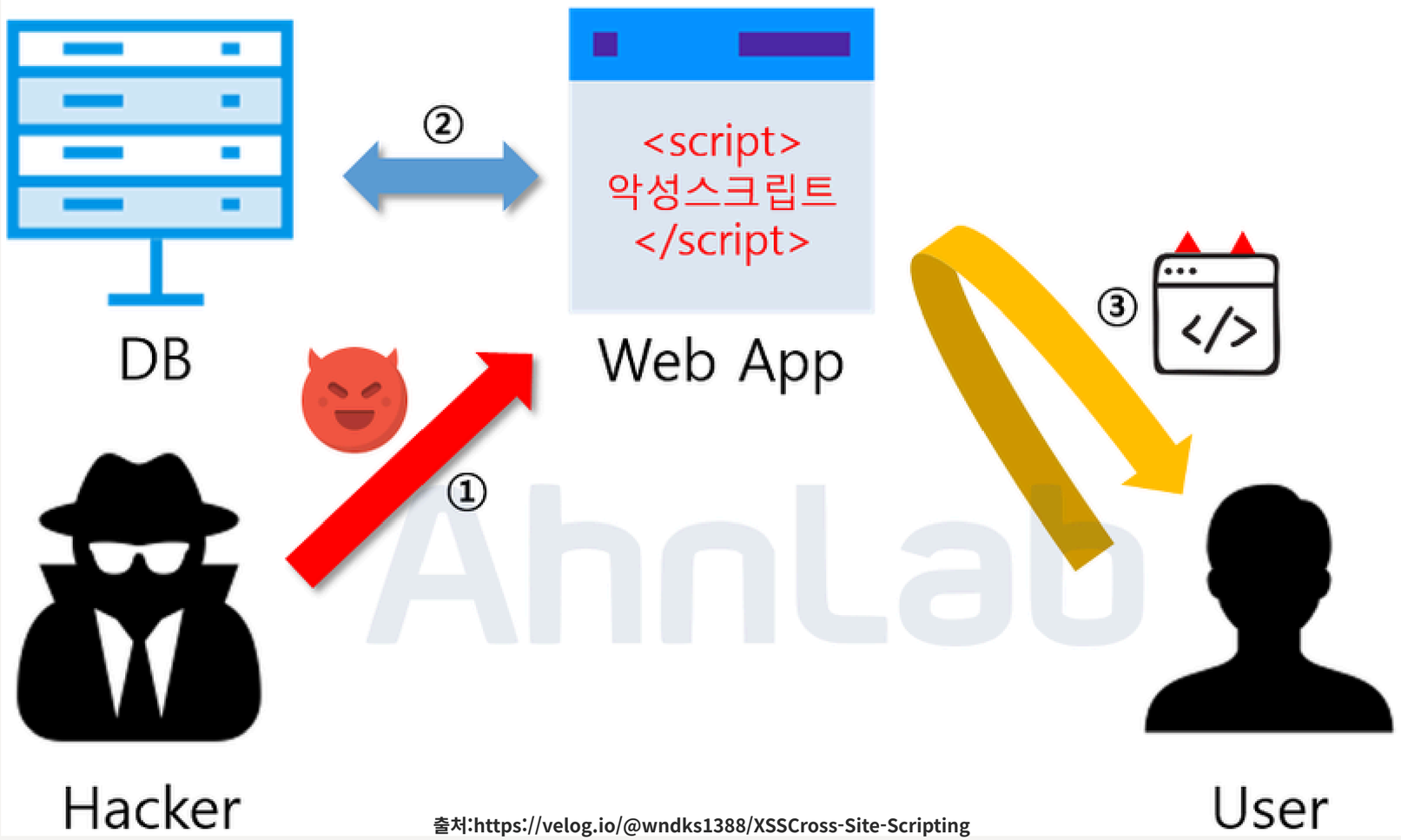
XSS의 세가지 주요 유형 - Stored XSS

Stored XSS (영구 저장형)
공격자가 악성 스크립트를 서버(예: 데이터베이스, 로그, 게시판 등)에 영구적으로 저장해 두고, 그 저장된 콘텐츠를 다른 사용자가 열람할 때 브라우저가 해당 스크립트를 실행하는 형태.

악성 스크립트가 저장된 게시글 등을 열람한 사용자들은 악성 스크립트가 작동함에 따라 쿠키나 세션 등의 민감 정보를 탈취당하거나 다른 웹사이트로 리다이렉션될 수도 있다.

발생 시나리오(흐름)
공격자 → 악성 입력(게시글/댓글/프로필 등) → 서버에 저장 → 다른 사용자가 해당 콘텐츠 열람 → 브라우저에서 스크립트 실행 → 세션 탈취·UI 변조·피싱 등 발생

XSS의 세가지 주요 유형 - Stored XSS



출처:https://velog.io/@wndks1388/XSSCross-Site-Scripting

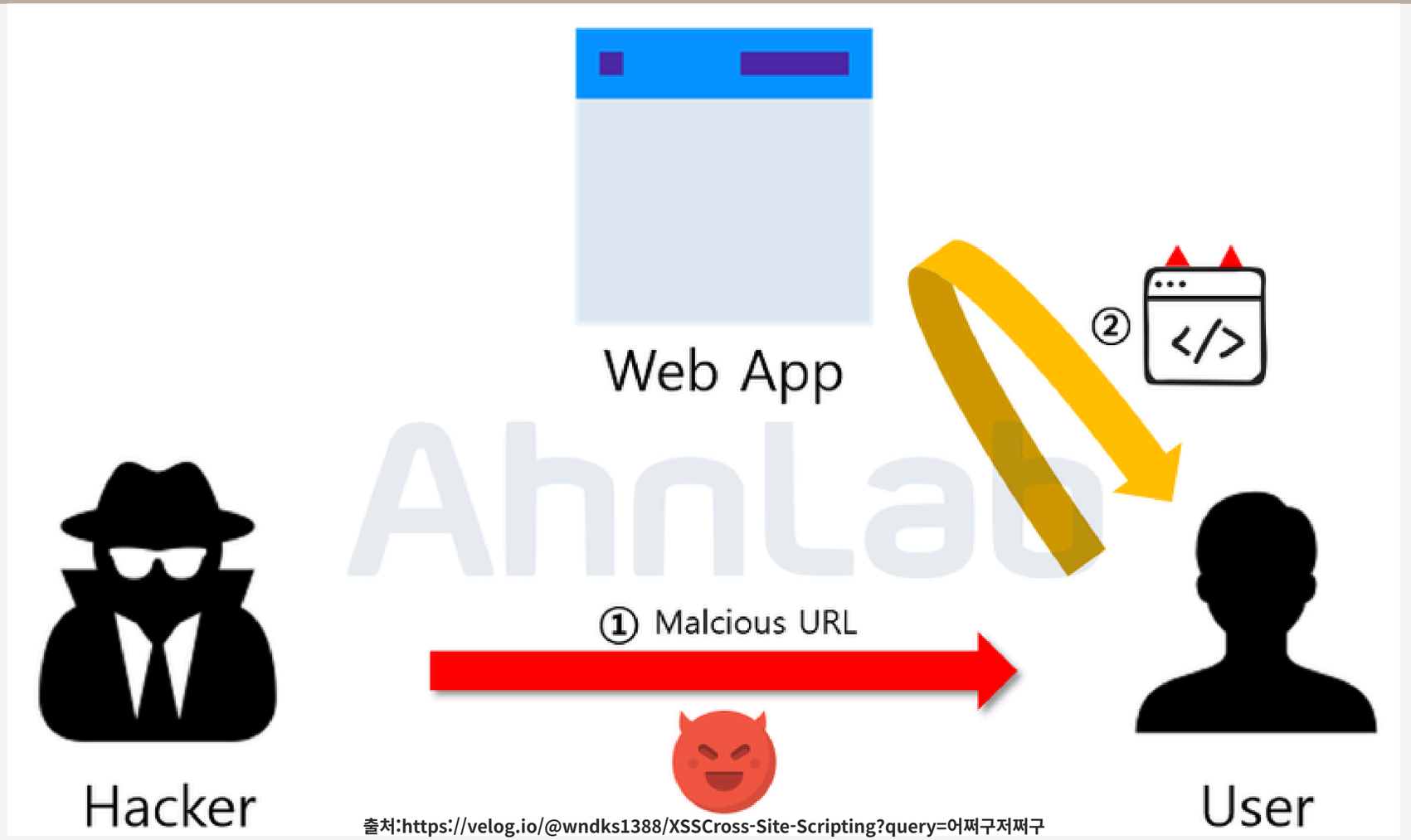
XSS의 세가지 주요 유형 - Reflected XSS

Reflected XSS (반사형)
공격자가 조작한 입력(주로 URL 쿼리스트링, 폼 파라미터 등)이 서버 응답에 그대로 포함되어 사용자 브라우저에서 즉시 실행되는 형태.

취약한 웹 사이트에 악성 스크립트를 실행하는 URL을 직접 사용자가 누르게 만든다.
URL 링크를 누르게 되면 자동적으로 악성 스크립트가 실행되게 된다.

발생 시나리오(흐름)
공격자 → 조작된 URL(또는 폼) 생성 → 피해자 클릭 → 서버가 입력을 이스케이프 없이 응답에 포함 → 피해자 브라우저에서 스크립트 실행

XSS의 세가지 주요 유형 - Reflected XSS



출처:https://velog.io/@wndks1388/XSSCross-Site-Scripting?query=어쩌구저쩌구

XSS의 세가지 주요 유형 - DOM-based XSS

DOM-based XSS (클라이언트 측 XSS)
서버 응답 자체에는 악성 스크립트가 포함되지 않지만, 클라이언트(브라우저) 자바스크립트가 DOM을 조작하는 과정에서 사용자 입력(location.hash, location.search, document.referrer, postMessage 등)을 안전하게 처리하지 않아 발생.

DOM based XSS공격은 악성 스크립트가 포함된 URL을 일반 사용자가 클릭하였을 때, 웹 서버에서 받아온 페이지를 클라이언트 브라우저에서 DOM을 구성하는 과정에서 일어난다.

발생 시나리오(흐름)
공격자 → 조작된 URL(예: 해시값) 또는 악성 메시지 → 브라우저에서 클라이언트 스크립트가 해당 값을 읽어 innerHTML 등으로 DOM에 삽입 → 삽입된 스크립트 실행

XSS의 세가지 주요 유형 - DOM-based XSS

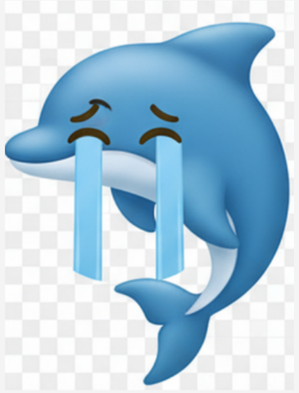
출처:https://velog.io/@wndks1388/XSSCross-Site-Scripting?query=어쩌구저쩌구

XSS 방어 대책

- 출력 인코딩(콘텍스트별 인코딩)을 기본으로 적용
- 입력 검증은 화이트리스트 우선 적용
- 템플릿 엔진의 자동 이스케이프 활용
- DOM 삽입 시 안전한 API(textContent 등) 사용 또는 신뢰 가능한 sanitizer 사용
- 콘텐츠 시큐리티 정책(CSP) 도입
- 민감 쿠키 HttpOnly/Secure/SameSite 설정

사용자가 입력한 데이터는 어떠한 경우에도 브라우저가 코드로 해석하도록 방치해서는 안돼!!

SQL Injection



SQL Injection
외부에서 입력된 데이터가 데이터베이스 쿼리의 문법·의미에 영향을
미치도록 처리될 때 발생하는 취약점.

공격자는 이를 이용해 본래 의도하지 않은 쿼리를 실행하게 하여, 데이터베이스의 민감정보를 빼내거나 데이터를 변조·삭제할 수 있습니다.

단순한 텍스트 입력이 데이터베이스 명령어로 '변신'하는 문제라고 이해하시면 됩니다.

사실 이것도 엄청 많지만.. 분량조절을 위해 스르륵 넘기겠습니다.

SQL Injection예시 (로그인)

SELECT * FROM users WHERE username = '${username}' AND password = '${password}

만약 username이 admin' -- 이라면, 어떤 쿼리가 완성될까요 ?

SQL Injection예시 (로그인)

SELECT * FROM users WHERE username = '${username}' AND password = '${password}
만약 username이 admin' -- 이라면, 어떤 쿼리가 완성될까요 ?

SELECT * FROM users WHERE username = 'admin'-- ' AND password = '${password}
MYSQL에서 -- 뒤에 부분은 주석으로 간주됩니다. 따라서

SELECT * FROM users WHERE username = 'admin'
이 쿼리로 변신해버립니다 !! 당연히 이러면, 비밀번호에 뭘 넣든, 관리자 계정으로 로그인이 되겠죠 ?

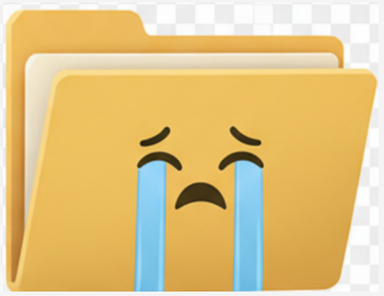
SQL Injection 방어 대책

- DB 계정은 최소 권한만 부여. 애플리케이션은 읽기/쓰기/관리 권한을 잘 나눔.
- 쿼리 구조와 값(파라미터)을 분리 → 값이 SQL 문법으로 해석되지 않음 (ORM 사용)
- ;로 여러 SQL 문을 한 번에 실행할 수 있으면 인젝션의 강력한 수단이 됨.
- 따라서 DB 드라이버/클라이언트에서 다중 스테이트먼트 기능을 비활성화.
- 가능한 경우 허용되는 값의 패턴만 받음(정수 ID, 이메일 형식 검증 등).
- DB 에러(스택·쿼리 등)를 사용자에게 그대로 노출하지 않기

이전과 마찬가지로, 서버는 사용자가 입력한 데이터는 어떠한 경우에도 의심하기!!!

Path(Directory) Traversal

Path(Directory) Traversal
애플리케이션이 파일 경로를 구성할 때 사용자 입력을 신뢰하여,
상대경로(../)나 인코딩을 이용해 원래 접근하면 안 되는 파일을 읽거나 쓰는 문제.



설정 파일이나 인증키, 사용자 데이터베이스 같은 민감 정보가 노출될 수 있으며, 경우에 따라서는 파일 덮어쓰기나 악성 파일 업로드로 이어져 원격 코드 실행까지 가능

쉽게 말씀드리자면, 서버의 디렉토리를 마구마구 여행하고 횡단(Traversal)하는 이슈입니다.

Path(Directory) Traversal 예시

```
app.get('/download/:filename', (req, res) => {
  const filePath = path.join(__dirname, 'public', req.params.filename);
  res.sendFile(filePath);
});
```

C:\Server\app.js 에서 실행중이며, C:\Server\public에 파일들이 있다고 가정합시다.

req.params.filename이 ../app.js라면 어떻게 될까요 ?

Path(Directory) Traversal 예시

```
app.get('/download/:filename', (req, res) => {
  const filePath = path.join(__dirname, 'public', req.params.filename);
  res.sendFile(filePath);
});
```

C:\Server\app.js 에서 실행중이며, C:\Server\public에 파일들이 있다고 가정합시다.

req.params.filename이 ../app.js라면 어떻게 될까요 ?

안녕하세요 ㅎㅎ 여기 제 서버코드입니다 ㅎㅎ 잘 쓰세요~ 가 됩니다.

Path(Directory) Traversal 방어 대책

- 사용자 입력을 파일 경로로 직접 사용하지 않는다.
- 가능한 경우 ID·토큰을 통해 서버 내부 매핑을 사용한다.
- 경로 정규화(canonicalization) → base directory 검사
- 사용자 입력을 정규화한 후 미리 허용한 루트(baseDir) 내부인지 확인한다.
- 화이트리스트 + 확장자·매직바이트 검사
- 허용되는 파일 타입·확장자·파일 헤더(매직바이트)만 허용한다.

이전과 마찬가지로, 서버는 사용자가 입력한 데이터는 어떠한 경우에도 의심하기!!!

시연을 통해 알아보자

모든 소스코드는 AI의 도움을 받아 작성했고,
로컬호스트에서 진행합니다.

아무 사이트에 막 시도해보시고 그러면 깜빵 갑니다.