WhiteHat School 1기

HTML, CSS, JS 취약점 Write-up

Team: 과부화

Writer: 정진교

취약점

HTML 보안 취약점

1. 크로스 사이트 스크립팅 (XSS):

- 크로스 사이트 스크립팅,즉 사이트 간의 스크립팅을 뜻하며 사이트 안의 게시물 또는 댓글에 해커가 작성한 '스크립트'를 삽입시켜서 사용자의 권한, 계정을 가로챌 수 있다.
- 사용자로부터의 입력을 적절하게 처리하지 않고 출력하는 경우 악의적인 스크립트가 실행될 수 있다. 이는 HTML에서 JavaScript 코드가 실행되는 방식으로 발생할 수 있다.

```
<!-- 취약점이 있는 코드 예시 -->
<p>Welcome, <%= user.name %></p>
```

- 아주 간단한 코드 예시

2. XSS의 종류

3. CSRF

- 웹 보안 공격 중 하나로, 인증된 사용자의 권한을 이용하여 악의적인 요청을 실행시키는 공격이다. 이 공격은 사용자가 이미 인증된 상태에서 악의적인 웹사이트를 방문하거나 특정 조작된 링크를 클릭할 때 발생할 수 있다.

3-1. CSRF 공격 흐름

3-1-1. 사용자가 웹 애플리케이션에 로그인한다.

3-1-2. 사용자는 다른 탭에서 악성 웹사이트를 방문하거나, 악의적인 이메일을 클릭하는 등의 방식으로 공격자의 제어 아래에 있는 페이지에 접근한다.

3-1-3. 악성 페이지에서는 특정 조작된 요청을 생성하고, 이를 공격 대상자의 웹 애플리케이션으로 보낸다.

3-1-4. 브라우저는 공격 대상자의 인증 쿠키를 자동으로 요청에 포함시켜 보낸다.

3-1-5. 웹 애플리케이션은 이 요청을 받고, 사용자가 이미 인증되었다고 판단하여 해당 요청을 처리한다.

4. SSRF

- SSRF(서버 사이드 요청 위조, Server-Side Request Forgery)는 공격자가 웹 애플리케이션 서버에서 다른 서버로의 요청을 위조하는 공격이다. 이는 주로 웹 애플리케이션에서 외부 리소스에 대한 요청을 수행할 때 발생하며, 공격자는 악의적으로 서버의 내부 리소스에 접근하거나 다른 외부 서버에 요청을 보낼 수 있다

- SSRF는 주로 URL 입력 값이 사용되는 경우에서 발생할 수 있다. 웹 애플리케이션에서 사용자로부터 입력받은 URL을 이용하여 다른 서버로의 요청을 만들 때, 공격자는 임의의 URL을 입력할 수 있다.

5. CSS

CSS는 주로 스타일을 정의하는 데 사용되는 스타일 시트 언어이며, 일반적으로 직접적인 보안 취약점을 가지지는 않는다. 그러나 CSS를 사용할 때 발생할 수 있는 몇 가지 보안 관련 주의사항이 있다.

스타일 시트 공격:

- 악의적인 사용자가 스타일 시트를 이용하여 페이지의 레이아웃을 손상시키거나 가상의 컨텐츠를 삽입하는 등의 공격이 있을 수 있다.

```
body { display: none; }
```

스타일 시트 인젝션:

- 사용자 입력이 스타일 시트로 직접 삽입되는 경우, 스타일 시트 인젝션이 발생할 수 있다. 이로 인해 스타일이 원하지 않는 방식으로 적용될 수 있다.

```
<link rel="stylesheet" type="text/css" href="style.css?color=<script>alert('XSS');</script>">
```

스타일 시트 도용 (CSS Hijacking):

- 공격자가 스타일 시트를 조작하여, 웹 페이지의 외관이나 레이아웃을 변경하는 CSS 도용이 발생할 수 있다.

```
body { background-color: pink !important; }
```

5-1. CSS Injection

CSS 인젝션은 주로 웹 애플리케이션에서 사용되는 스타일 시트에 악의적인 코드를 삽입하는 공격을 나타낸다. 공격자가 스타일 시트에 액세스하여 스타일을 변조하거나, 사용자가 제공한 입력을 통해 스타일을 조작하는 방식으로 이루어진다.

```
<style>
  body {
    background-color: <?= $_GET['color'] ?>;
  }
</style>
```

위 코드에서 $_GET['color']는 사용자가 제공한 URL 매개변수를 통해 받은 값이다.

예) '?color=red; alert('CSS Injection!');' 와 같은 값이 전달된다면
'alert('CSS Injection!');' 코드가 실행될 수 있다.

6. JS Injection

JavaScript 인젝션은 웹 애플리케이션에서 JavaScript 코드를 악의적으로 삽입하여 실행하려는 공격을 나타낸다. 이는 주로 사용자가 제공한 입력을 통해 발생하며, 공격자가 애플리케이션에서 예상치 못한 동작을 유발하거나 중요한 정보를 탈취하려는 시도로 이루어진다.

```
<script>
  var username = '<?= $_GET['username'] ?>';
  alert('Welcome, ' + username + '!');
</script>
```

$_GET['username']는 사용자가 제공한 URL 매개변수를 통해 받은 값이다.

공격자는 URL을 통해 임의의 JavaScript 코드를 삽입할 수 있다.

예) '?username='); alert('XSS');' 값이 전달된다면 '); alert('XSS');'
코드가 실행될 수 있다.

보안대책

XSS 관련 보안대책

입력 데이터의 검증 및 필터링:

- 사용자가 입력한 데이터를 검증하고 필터링하여 안전하지 않은 문자나 스크립트를 방지한다. 입력값에 특수 문자나 스크립트를 필터링하거나 이스케이프하여 처리한다.

쿠키 보안 설정:

- 쿠키에 중요한 정보를 저장하지 않거나, 보안 플래그(secure, HttpOnly)를 사용하여 스크립트로부터의 접근을 방지한다.

Content Security Policy (CSP) 사용:

- CSP를 통해 웹 페이지에서 로드할 수 있는 리소스의 출처를 제한하여 XSS 공격을 방지한다.

HTTP 헤더 설정:

- X-Content-Type-Options 헤더를 설정하여 브라우저가 MIME 타입을 자동 감지하도록 하거나, X-Content-Security-Policy 헤더를 사용하여 XSS 공격을 방지할 수 있다.

클라이언트 사이드 보안:

- 사용자 입력을 표시할 때 JavaScript Escape 함수를 사용하여 스크립트가 실행되지 않도록 한다.

세션 관리 보안:

- 세션 식별자를 안전하게 관리하고, 민감한 작업을 수행하기 전에 사용자를 재인증하는 등의 보안 조치를 적용한다.

보안 업데이트 및 취약성 관리:

- 사용하는 라이브러리, 프레임워크, 서버 소프트웨어 등을 최신 상태로 유지하고 보안 업데이트를 적용한다.

필터링 방어책의 회피 방지:

- 입력 값 이스케이프 (Output Encoding):
  -- 사용자 입력이 출력될 때, 특수 문자를 이스케이프하여 브라우저가 해당 문자를 스크립트로 해석하지 못하도록 만든다.

- Whitelisting:
  -- 허용되는 문자나 패턴을 명시적으로 정의하여, 허용된 것 이외의 모든 것을 거부하는 방식.
  예)특정 HTML 태그나 속성을 허용하고 나머지는 거부하는 방식

- Content Security Policy (CSP):
  -- CSP를 사용하여 웹 페이지에서 로드할 수 있는 리소스의 출처를 명시적으로 지정한다. 스크립트 실행을 허용하는 출처를 제한함으로써 XSS 공격을 방지한다.

- JavaScript Escape Functions:
  -- JS에서 특수 문자를 이스케이프하는 함수를 사용하여 출력값을 처리한다. encodeURIComponent(), encodeURI(), escape(), JSON.stringify() 등이 해당된다.

- HTTP 헤더 설정:
  -- Content Security Policy (CSP) 헤더를 사용하거나 X-XSS-Protection 헤더를 설정하여 브라우저 내장 XSS 필터를 활성 가능하다.

- DOM 기반 XSS 방어:
  -- 클라이언트 측에서 DOM 조작을 통한 XSS를 방지하기 위해 안전한 DOM 조작 방법을 사용한다. document.createElement(), element.textContent 등을 활용할 수 있다.

웹 어플리케이션 방화벽 (WAF):
- 네트워크를 통해 들어오는 HTTP 트래픽을 검사하여 악성 패턴을 차단한다. 몇몇 WAF는 XSS 공격에 대한 패턴을 감지하고 차단한다.

정규 표현식 (Regular Expressions):
- 입력값에 대한 검증에 정규 표현식을 사용하여 특수 문자를 필터링하거나 특정 패턴을 차단할 수 있다.

CSRF 관련 보안대책

CSRF 토큰 사용:
- 웹 애플리케이션은 사용자의 세션과 관련된 고유한 CSRF 토큰을 생성하고 이를 웹 페이지의 폼이나 AJAX 요청에 포함시킨다. 악성 웹사이트는 이 토큰을 알지 못하기 때문에 CSRF 공격이 어려워진다.

SameSite 쿠키 속성 사용:
- SameSite 쿠키 속성을 Strict 또는 Lax로 설정하여, 외부 사이트에서의 요청에 대한 쿠키 전송을 제한함으로써 CSRF 공격을 방지할 수 있다.

Referrer 검증:
- 웹 애플리케이션은 HTTP Referer 헤더를 검증하여 요청이 예상된 도메인에서 왔는지 확인할 수 있다.

Anti-CSRF 라이브러리 사용:

- 다양한 프레임워크나 라이브러리에서는 CSRF 공격을 방지하기 위한 보안 기능을 제공한다.

보안 정책 및 권한 관리:

- 웹 애플리케이션에서는 민감한 작업에 대한 권한 검사를 강화하여, 특정 작업에 대한 권한이 있는 사용자만 해당 작업을 수행하도록 합니다.

SSRF 관련 보안대책

입력 값의 검증과 필터링:

- 사용자로부터 입력받은 URL 또는 리소스 위치를 엄격하게 검증하고, 허용된 도메인 또는 IP 주소만 허용한다. URL을 검증할 때는 정규 표현식 등을 사용하여 안전한 형식인지 확인한다.

화이트리스트 방식 사용:

- 허용된 목록에 속하지 않은 리소스에 대한 요청을 차단하는 화이트리스트 방식을 채택한다. 허용할 도메인 또는 IP 주소 목록을 정의하고, 이외의 요청을 차단한다.

제한된 프록시 설정:

- 웹 애플리케이션 서버가 외부 리소스에 대한 요청을 할 때 사용하는 프록시 설정을 최소화하고, 필요한 경우에만 프록시 서버를 사용하도록 한다.

서버 구성 및 리소스 접근 권한 관리:

- 서버에서 다른 서버로의 통신이 필요한 경우, 최소 권한의 원칙을 따르고 필요한 권한만을 부여한다. 필요하지 않은 서비스나 포트는 비활성화한다.

로컬 IP 루프백 차단:

- 서버에서 자체적으로 생성된 요청이 내부 리소스에 액세스할 수 없도록 로컬 IP 주소나 루프백 주소에 대한 요청을 차단한다.

서버에서 URL을 하드코딩하지 않기:

- 서버 측 코드에서 URL을 하드코딩하지 않고, 동적으로 생성하거나 설정 파일에서 가져오도록 구현한다.

Content Security Policy (CSP):

- CSP를 사용하여 외부 도메인과의 상호작용을 제한하고 허용된 도메인만을 지정한다.

네트워크 레벨 방어책 적용:

- 방화벽 등의 네트워크 레벨에서 외부로의 트래픽을 제한하고, 내부 서버에 대한 외부에서의 직접적인 접근을 차단한다.

스타일 시트 공격 방어책:

- 사용자 입력을 적절하게 검증하고, 사용자에게 스타일을 제어할 수 있는 기능을 제한해야 한다. 또한, 사용자가 제공한 스타일을 적용하기 전에 검증해야 한다.

스타일 시트 인젝션 방어책:

- 사용자 입력값을 이스케이핑하여 스타일 시트로 삽입되는 것을 방지한다.

스타일 시트 도용 방어책:

- 스타일 시트에 대한 액세스 권한을 엄격히 제한하고, 스타일 시트를 안전한 방식으로 로드한다.

CSS, JS 관련 보안책

CORS (Cross-Origin Resource Sharing) 설정:

- 웹 애플리케이션에서는 필요한 경우에만 CORS를 허용하도록 설정하여, 악의적인 도메인에서의 스크립트 실행을 방지할 수 있다.

Content Security Policy (CSP) 설정:

- CSP를 사용하여 어떤 리소스가 로드될 수 있는지를 명시적으로 정의하여 외부 스크립트의 실행을 제어한다.