Engineer

Content Security Policy

en lista över varifrån en sida får läsa in saker

Vad det är

Content Security Policy är en uppsättning regler som en webbplats lämnar till webbläsaren om vad dess sidor får köra, hämta och skicka, och som webbläsaren sedan upprätthåller för webbplatsens räkning. Namnet förkortas nästan alltid CSP. Webbplatsen skickar reglerna i svarsheadern `Content-Security-Policy`, eller skriver in dem i sidan i ett `meta`-element, och från det ögonblicket stoppas varje skript, stilmall, bild, typsnitt, ram eller anslutning som reglerna inte tillåter innan det hinner laddas.

En policy är en kort sträng av direktiv åtskilda med semikolon. Varje direktiv namnger ett slags resurs eller beteende och de källor som det godtar. En strikt policy ser ut så här, med ett direktiv per rad för läsbarhetens skull, fast den i trafiken är ett enda headervärde:

```
default-src 'self';
script-src 'nonce-q8Zt3VwR1mKx7NpL2cYfJA==' 'strict-dynamic';
object-src 'none';
base-uri 'none';
form-action 'self';
frame-ancestors 'none'
```

Vokabulären är liten. `'self'` betyder sidans eget ursprung. En källa med värdnamn, som `https://static.example.com`, eller en protokolldel som `https:` vidgar det. `'none'` tillåter ingenting. En nonce är ett slumpvärde som servern tar fram för ett enda svar och skriver både i policyn och på varje skriptelement som den går i god för. En hash är det kryptografiska avtrycket av ett inline-skripts exakta text, så att just det skriptet och inget annat får köras. Två nyckelord, `'unsafe-inline'` och `'unsafe-eval'`, slår av skydden igen, och deras namn är en rättvis varning.

`default-src` fungerar som reserv för de hämtningsdirektiv som en policy utelämnar, t.ex. `script-src`, `style-src`, `img-src` och `connect-src`. Det gäller bara för dem. Direktiv som styr beteende snarare än hämtning, bl.a. `base-uri`, `form-action` och `frame-ancestors`, förblir obegränsade om de inte skrivs ut.

En policy kan också köras i ett bevakande läge. Skickad som `Content-Security-Policy-Report-Only` blockerar den ingenting och rapporterar i stället varje överträdelse till en mottagare som namnges med `report-to`, som använder webbläsarens Reporting API. Det äldre direktivet `report-uri` är föråldrat till förmån för det. Den aktuella texten i standarden är Content Security Policy Level 3, ett arbetsutkast (Working Draft) från W3C:s arbetsgrupp Web Application Security.

Använd begreppet i ett beslut

Använd ”Vad är Content Security Policy?” i ett konkret beslut. Vi kan knyta begreppet till systemet ni bygger eller köper. Konton, lösenord och behörigheter ordnade så att alla har exakt den åtkomst deras arbete kräver, och de som slutar förlorar den samma dag.

Se våra tjänster Hör av er →

Granska antagandena bakom ordet

Använd förklaringen och det länkade arbetet för att formulera frågor till projektet. Vi kan undersöka antagandena tillsammans innan ett verktyg väljs. Skrev 1 000 RBAC-regler som säkrar Linux-servrar — 1,000 åtkomstregler, skrivna för att täppa till en säkerhetslucka

Skrev 1 000 RBAC-regler som säkrar Linux-servrar Hör av er →

Varför det spelar roll

Attacken som CSP byggdes mot är cross-site scripting, ofta förkortat XSS. Någon lyckas få in text som de själva har valt i en sida, via ett kommentarsfält, en sökterm som ekas tillbaka i HTML-koden eller ett kapat beroende, och webbläsaren kör den som skript med samma befogenheter som webbplatsens egen kod: den kan läsa sidan, agera som den inloggade användaren och skicka det den hittar vart som helst. Det första försvaret är att koda utdata korrekt, så kallad escaping, och det måste vara rätt i varje mall, på varje rad, varje gång. CSP är det andra försvaret. När ett misstag i kodningen ändå slinker igenom måste det injicerade skriptet fortfarande ta sig förbi policyn, och en policy som bara kör skript som bär just det här svarets nonce avvisar det.

Den begränsar också vad ett skript som faktiskt körs kan nå. `connect-src` avgör vilka värdar skript får tala med, vilket begränsar vart en skimmer, ett skript som stjäl kortuppgifter, kan skicka det den samlar in. `frame-ancestors` avgör vilka webbplatser som får visa sidan i en ram, standardförsvaret mot clickjacking, där en sida läggs över en annan så att ett klick hamnar någonstans där användaren

aldrig tänkte klicka. `form-action` avgör vart formulär får skickas, och `base-uri` hindrar ett injicerat `base`-element från att i tysthet styra om varje relativ adress på sidan.

Det lönar sig att vara exakt om vem CSP skyddar. Policyn körs i webbläsaren, så den skyddar den som läser sidan mot innehåll som en angripare har planterat där. Den gör ingenting för servern mot samma person, som kan ändra eller ta bort headern i sin egen webbläsare på några sekunder. En policy är ett skäl att lita mindre på sidan, aldrig ett skäl att lita mer på förfrågan.

Kostnaden är mest inventering. En policy tvingar ett team att känna till varje ursprung som dess sidor laddar från, varje inline-skript och varje händelsehanterare som är skriven som attribut. En ny webbplats kan byggas mot en strikt policy från första commit för väldigt lite. En äldre webbplats med inline-hanterare, en tagghanterare och flera widgetar betalar först för uppstädningen, och den uppstädningen är oftast det verkliga projektet.

Så ser det ut i praktiken

Den här webbplatsen lever själv under en sådan policy. Sidan om hur vi arbetar säkert listar en strikt Content Security Policy, deklarerad i själva sidan, med `form-action 'none'`, eftersom webbplatsen inte skickar några formulär. Varje HTML-sida anger `script-src 'none'`: varken JavaScript eller någon service worker körs. Webbplatsen gör inga anrop till tredje part; allt som en sida behöver kommer från webbplatsens egen värddator, så det finns väldigt lite att tillåta.

En applikation med inloggade användare behöver mer än en kort lista över tillåtna källor. På en plattform som lagrar yrkes- och organisationsdata sätter middlewaren i Next.js en Content-Security-Policy med en nonce per anrop och `strict-dynamic`, tillsammans med HSTS och SameSite-kakor. En ny nonce i varje svar betyder att bara de skriptelement som servern stämplade i just det svaret körs. Med `strict-dynamic` i policyn får ett skript som nonce-värdet litar på själv ladda fler skript, så policyn behöver inte namnge varje värddator som ett JavaScript-paket hämtar sina delar från. Samma sida är tydlig med var webbläsarens del slutar. Arbetsantagandet där är att klienten är fientlig, och policyn är därför ett lager av flera: den skyddar användarna mot injicerad kod, medan behörigheten avgörs någon annanstans: servern tittar på rättigheterna vid varje enskilt anrop, och spärrarna i frontenden räknas bara som UX.

Var det går fel

Policyn går i drift och halva webbplatsen slutar fungera. En strikt policy som skickas direkt till skarpt läge blockerar det inline-skript som ingen kom ihåg och spårningstaggen som någon lade till för två år sedan. Skicka den först som `Content-Security-Policy-Report-Only`, läs rapporterna tills de lugnar sig, tillåt eller ta bort varje källa som de nämner och gå först därefter över till skarpt läge. Report-only-headern måste vara en riktig header: specifikationen stöder den inte i ett `meta`-element.

Listan över tillåtna källor litar på en hel värddator. `script-src https://cdn.example.com` tillåter varje fil på den värddatorn, inklusive gamla versioner av bibliotek och ändpunkter som ekar tillbaka anroparens indata som skript. Specifikationen säger själv att det är en utmaning att driftsätta en policy som fungerar mot cross-site scripting, och rekommenderar det som den kallar en Strict CSP: `script-src` byggt av nonces eller hashar med `'strict-dynamic'`, plus `base-uri` satt till `'self'` eller `'none'`.

`'unsafe-inline'` **antas tillåta alla inline-skript.** Så snart en nonce eller en hash finns i samma källista ignorerar webbläsare som förstår dem `'unsafe-inline'`. Regeln låter en policy behålla `'unsafe-inline'` som reserv för webbläsare som är för gamla för att känna till nonces, medan dagens webbläsare tillämpar den strikta listan, och den överraskar ett team som lägger till en nonce på ett skript och sedan ser sina andra inline-skript blockeras.

Nonce-värdet slutar vara en hemlighet. Specifikationen kräver ett nytt värde varje gång en policy skickas och ber om minst 128 bitar från en kryptografiskt säker slumpgenerator. En nonce som är hårdkodad i en mall, eller en hel sida som lagras med sin nonce i en delad cache och serveras till alla, ger en angripare precis det värde som behövs för att stämpla ett eget skript.

En policy i ett `meta`-element gör mindre än samma policy i en header. Webbläsaren ignorerar `frame-ancestors`, `report-uri` och `sandbox` när de kommer den vägen, och policyn styr bara det som kommer efter elementet i dokumentet, så allt ovanför det i `head` laddas okontrollerat. En webbplats som inte rår över sina svarsheaders får ändå ett verkligt skydd av en policy i `meta`, placerad så tidigt i `head` som det går, men skydd mot inramning kräver headern.

En andra policy stramar åt i stället för att lätta. Ett svar kan bära flera policyer, från ett ramverk, en proxy och ett CDN, och en resurs måste klara var och en av dem. En tillagd policy kan bara begränsa, så en tillåtande header som läggs till längre ned i kedjan för att rätta en blockerad förfrågan gör ingenting, och rättelsen måste göras i den policy som blockerade.

Läs mer

- Content Security Policy Level 3 — själva standarden, med rekommendationen om Strict CSP och ett avsnitt om hur nonces blir stulna.
- HTML Standard: the meta element — pragmat som tillämpar en policy som deklareras i sidan, direktiven som det släpper och varför resurser som hämtas före det kan slinka förbi.
- Reporting API — mekanismen för ändpunkter bakom `report-to`, för att läsa överträdelser innan policyn skärps.
- Trusted Types — standarden vid sidan av CSP som direktivet `require-trusted-types-for` slår på, riktad mot skript som injiceras via DOM.

Ordlista

Välj utifrån era faktiska förutsättningar

Ta med driftsättningsvägen, driftsriskerna och dem som har jour. Vi kan avgränsa en förändring med kontroller, en väg tillbaka och tydligt ansvar.

Se vad det kostar Hör av er →

Gör begreppen praktiska

Fråga hur ”Vad är Content Security Policy?” gäller i er organisation. Ett konkret exempel från ert nuvarande system räcker för att börja samtalet.

Se våra tjänster

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

Ta med dig den här sidan

Visitkort

Omsorgsfullt skapad med kärlek av D.

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

Upphovsrätt © 2025 – nu · Engineer ApS · Alla rättigheter förbehållna.

Utskrivet från https://engineer.company/sv/glossary/csp/