Engineer

HTTPS

🔑 de samme forespørslene, over en kryptert forbindelse

Hva det er

HTTPS er HTTP sendt gjennom en TLS-forbindelse. Forespørslene og svarene er de samme meldingene som før, med de samme metodene, statuskodene og headerne; det som endres, er forbindelsen under dem, som nå er kryptert og autentisert. URI-skjemaet `https` er definert i RFC 9110, spesifikasjonen av HTTP-semantikken fra juni 2022. Den tok over etter RFC 2818, det korte dokumentet fra mai 2000 som først skrev ned HTTP over TLS. Standard nettverksport er 443, der ukryptert HTTP bruker 80.

RFC 9110 knytter to regler til skjemaet. En klient skal ubetinget kontrollere at serverens identitet passer til verten i adressen, og i praksis betyr det at sertifikatet må dekke navnet; en forbindelse til feil navn har feilet, uansett hvor sterk krypteringen er. Og `http://example.com/` og `https://example.com/` er to forskjellige opphav, origins på engelsk, med hvert sitt navnerom. For protokollen deler de et vertsnavn og ingenting annet.

HTTPS er ikke bundet til én versjon av HTTP. HTTP/1.1 og HTTP/2 går over TLS på TCP. HTTP/3 går over QUIC, som bærer håndtrykket i TLS 1.3 inne i selve transporten, og RFC 9114 utelukker å bruke den for å nå en ukryptert `http://` -server direkte. Den nyeste versjonen kommer altså med HTTPS eller ikke i det hele tatt.

Hva krypteringen dekker, er lettest å se med en adresse. For `https://shop.example/orders/1234?ref=mail` er stien, spørrestrengen, hver header, informasjonskapsler og `Authorization` medregnet, og begge meldingenes body kryptert. Nettverket mellom den besøkende og serveren ser fortsatt serverens IP-adresse og port, når trafikken går og hvor stor den er, og i de fleste forbindelser også vertsnavnet `shop.example` : nettleseren slo det opp i DNS, og

klienten oppgir det i den første TLS-meldingen, slik at én adresse kan betjene mange nettsteder. HTTPS skjuler hva som ble spurt om, men ikke hvem som ble spurt.

Bruk begrepet i en beslutning

Bruk «Hva er HTTPS?» i en konkret beslutning. Vi kan knytte begrepet til systemet dere bygger eller kjøper. Kontoer, passord og tilganger ordnet slik at folk har nøyaktig den tilgangen jobben krever, og de som slutter, mister den samme dag.

Se tjenestene våre Ta kontakt →

Undersøk antakelsene bak ordet

Bruk forklaringen og det tilknyttede arbeidet til å formulere spørsmål for prosjektet. Vi kan undersøke antakelsene sammen før et verktøy velges. Skrev 1 000 RBAC-regler for å sikre Linux-servere — 1,000 tilgangsregler skrevet for å tette et sikkerhetshull

Skrev 1 000 RBAC-regler for å sikre Linux-servere Ta kontakt →

Hvorfor det er viktig

Det åpne spørsmålet for et offentlig nettsted er om noe fortsatt skal svare på `http://`, og nettleserne tar nå det meste av den avgjørelsen selv.

Den første spaken er sikker kontekst, et W3C-begrep for en side levert over HTTPS eller fra maskinen selv. Nettlesere forbeholder et voksende sett med funksjoner for sikre kontekster. En service worker er ett eksempel: `navigator.serviceWorker` er deklarert bare for sikre kontekster, så på en side lastet over ukryptert HTTP finnes egenskapen rett og slett ikke, og det gjør heller ingen funksjon bygd på den, med caching for bruk uten nett som den første. Spesifikasjonen for sikre kontekster anbefaler også at de som skriver nye spesifikasjoner, beskytter sensitive funksjoner på samme måte, så begrensningen er utgangspunktet for nytt arbeid og ikke et unntak vurdert fra sak til sak.

Den andre spaken er blandet innhold. Når en side selv er HTTPS, sorterer nettleseren hver av hentingene dens etter W3C-reglene for Mixed Content. Bilder, lyd og video som blir bedt om over `http://`, oppgraderes automatisk til `https://`, og de feiler hvis den adressen ikke svarer. Skript, stilark og dataforespørsler blokkeres blankt med en nettverksfeil. Én gjenglemt ukryptert adresse i en mal kan dermed ta ut en hel funksjon, ikke bare et bilde.

Da blir det svært lite igjen til port 80. En lytter som svarer på hver forespørsel med en permanent omdirigering til den samme stien på `https://`, fortjener fortsatt plassen sin, fordi folk skriver nakne navn og gamle lenker fortsatt sier `http://`. Sertifikatutstederens HTTP-utfordring kommer også inn der når et

sertifikat utstedes eller fornyes. HSTS lar så en nettleser som kommer tilbake, hoppe over selv omdirigeringen. Alt utover de jobbene på port 80 er en side servert uten garantiene ovenfor.

Loopback er det ene faste unntaket for vanlige nettsteder. Spesifikasjonen regner `localhost`, `127.0.0.1` og `::1` som potensielt pålitelige, så en side servert fra utviklerens egen maskin får funksjonene for sikre kontekster uten sertifikat. Den bekvemmeligheten for utviklere er kilden til en av feilene nedenfor.

Slik ser det ut i praksis

Selskapets eget nettsted følger den regelen. På produksjonsverten svarer Caddy på hver webforespørsel og fornyer HTTPS-sertifikatet selv, fra Let's Encrypt, så det er webserverens jobb og ikke en persons å holde sertifikatet gyldig. De samme sidene publiseres som en onion-tjeneste i Tor, og den adressen begynner med `http://` med vilje. En onion-adresse koder inn tjenestens offentlige identitetsnøkkel. Tor autentiserer tjenesten og krypterer tilkoblingen før HTTP utveksles, og derfor bruker dette speilet HTTP over Tor. Den samme siden lenker til Qualys SSL Labs, som gir den aktive HTTPS-konfigurasjonen en karakter for alle som spør.

Kundearbeid viser regelen om sikker kontekst fra den andre siden. For en plattform der brukerne ofte er sjøfolk og feltpersonell på mobilen, og ofte uten nett, leverte selskapet full støtte for Progressive Web App, med en service worker som cacher fonter, bilder, lyd, video og CDN-ressurser etter strategien CacheFirst. Fordi en service worker bare egentlig oppfører seg skikkelig over HTTPS, ble oppførselen uten nett og ved installasjon testet over HTTPS på ekte enheter i stedet for å bli tatt for gitt. Det samme behovet dukker opp i verktøyene til et polyglott monorepo med 578 oppgavemål, der go-task-laget selskapet skrev, legger en HTTPS-variant for testing av PWA og mobil ved siden av utviklingsmodusene native og Docker.

Hvor det går galt

Skript slutter å kjøre etter overgangen til HTTPS, og noen bilder forsvinner. Sidene inneholder fortsatt `http://`-adresser, hardkodet i en mal, lagret i innholdet i et CMS eller limt inn sammen med en snutt fra en tredjepart. Nettleseren blokkerer skriptene og stilarkene og oppgraderer bildene, som så brekker overalt der den andre verten mangler HTTPS. Nettleserkonsollen navngir

hver blokkerte adresse. Søk gjennom malene og det lagrede innholdet etter det gamle skjemaet, og skriv om hvert treff til `https://` eller til en sti relativ til nettstedet:

```
grep -rnE '(src|href)="http://' templates/ content/
```

En header `Content-Security-Policy: upgrade-insecure-requests` bærer sidene gjennom oppryddingen, men lar de lagrede adressene stå som de var.

Nettstedet omdirigerer til seg selv til nettleseren gir opp. Et CDN eller en lastbalanserer terminerer TLS og sender forespørselen videre til applikasjonen over ukryptert HTTP. Applikasjonen ser `http` og omdirigerer til `https`, proxyen sender det nye forsøket videre som `http` igjen, og løkken ender i feilen «for mange omdirigeringer». Den samme blindheten får applikasjonen til å bygge absolutte lenker som begynner med `http://`. Proxyen må sende med det opprinnelige skjemaet, i standardheaderen `Forwarded: proto=https` fra RFC 7239 eller i den eldre `X-Forwarded-Proto`, og applikasjonen bør bare tro på den headeren når den kommer fra proxyens egen adresse, siden enhver klient kan sende den. La så ett lag eie omdirigeringen, normalt det laget som har sertifikatet.

En informasjonskapsel for økten, satt over HTTPS, går i klartekst. RFC 9110 advarer om at informasjonskapsler ikke respekterer grensen den trekker mellom de to opphavene. Etter reglene i RFC 6265 blir en informasjonskapsel uten attributtet `Secure` også sendt med ukrypterte `http://`-forespørsler til den samme verten. Det holder at en besøkende skriver det nakne navnet, og like godt holder det med en angriper på nettverket som besvarer en hvilken som helst urelatert ukryptert forespørsel med en omdirigering til nettstedets `http://`-adresse. Sett `Secure` på hver informasjonskapsel nettstedet utsteder, og legg til HSTS, slik at nettleseren aldri sender den ukrypterte forespørselen i utgangspunktet.

Service workeren kjører på den bærbare maskinen og mangler på telefonen. Utvikleren åpner `http://localhost:3000`, en sikker kontekst, og alt virker. Telefonen på det samme nettverket åpner `http://192.168.1.20:3000`, som ikke er det, så `navigator.serviceWorker` er udefinert, et bibliotek som ser etter den, hopper over registreringen, og modusen uten nett er borte uten et ord. Server utviklingsbygget over HTTPS også. Caddy er ett eksempel på hvordan: den utsteder sertifikater for lokale navn og IP-adresser fra sin egen lokale

sertifikatutsteder og prøver å legge rotsertifikatet inn i maskinens tillitslager. Dokumentasjonen advarer om at steget ikke er garantert, og at det er administratorens jobb å få roten inn i hvert tillitslager, og en testtelefon må få roten installert for hånd.

Les mer

- RFC 9110, section 4.2.2: https URI Scheme — skjemaet, standardporten og regelen om at en klient må kontrollere serverens identitet.
- Secure Contexts — hvilke sider som regnes som sikre, hvorfor loopback er med, og rådet om å begrense nye funksjoner.
- Mixed Content — hvilke forespørsler fra en HTTPS-side som oppgraderes, og hvilke som blokkeres.
- RFC 7239: Forwarded HTTP Extension — standardheaderen en proxy bruker for å sende videre skjemaet den besøkende brukte.
- Automatic HTTPS — Caddys egen beskrivelse av sertifikatene, omdirigeringen fra HTTP og den lokale sertifikatutstederen.

Ordliste

Velg ut fra deres faktiske rammer

Ta med utrullingsløpet, driftsrisikoene og dem som har vakt. Vi kan avgrense en endring med kontroller, en vei tilbake og tydelig ansvar.

Se hva det koster Ta kontakt →

Gjør begrepene praktiske

Spør hvordan «Hva er HTTPS?» gjelder i organisasjonen deres. Et konkret eksempel fra dagens system er nok til å starte samtalen.

Se tjenestene våre

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

Ta med deg denne siden

Visittkort

Laget med kjærlighet av D.

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

Opphavsrett © 2025 – nå · Engineer ApS · Alle rettigheter forbeholdt.

Skrevet ut fra https://engineer.company/nb/glossary/https/