Engineer

HTTPS

🔑 samma begäranden, över en krypterad anslutning

Vad det är

HTTPS är HTTP som skickas genom en TLS-anslutning. Begäranden och svar är samma meddelanden som förut, med samma metoder, statuskoder och huvudfält; det som ändras är anslutningen under dem, som nu är krypterad och autentiserad. URI-schemat `https` definieras i RFC 9110, specifikationen för HTTP:s semantik från juni 2022. Den tog över efter RFC 2818, det korta dokumentet från maj 2000 som först skrev ner HTTP över TLS. Standardporten är 443, där okrypterad HTTP använder 80.

RFC 9110 knyter två regler till schemat. En klient måste kontrollera att serverns identitet stämmer godtagbart med värden i adressen, vilket i praktiken betyder att certifikatet måste täcka det namnet; en anslutning till fel namn har misslyckats, hur stark krypteringen än är. Och `http://example.com/` och `https://example.com/` är olika ursprung med var sin namnrymd. De delar ett värdnamn och, för protokollets del, ingenting annat.

HTTPS är inte bundet till en viss version av HTTP. HTTP/1.1 och HTTP/2 körs över TLS på TCP. HTTP/3 körs över QUIC, som bär handskakningen i TLS 1.3 inuti själva transporten, och RFC 9114 utesluter att den används för att nå en okrypterad `http://` -server direkt. Den nyaste versionen kommer alltså med HTTPS eller inte alls.

Vad krypteringen täcker syns lättast med en adress. För `https://shop.example/orders/1234?ref=mail` är sökvägen, frågesträngen, varje huvudfält inklusive kakor och `Authorization` , och båda meddelandenas kroppar krypterade. Nätverket mellan besökaren och servern ser fortfarande serverns IP-adress och port, trafikens tidpunkter och storlek, och i de flesta anslutningar

även värdnamnet `shop.example` : webbläsaren slog upp det i DNS, och klienten nämner det i sitt första TLS-meddelande så att en adress kan betjäna många webbplatser. HTTPS döljer vad någon bad om, men inte vem frågan ställdes till.

Använd begreppet i ett beslut

Använd ”Vad är HTTPS?” 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

Den aktuella frågan för en publik webbplats är om något fortfarande ska svara på `http://`, och webbläsarna fattar numera det mesta av det beslutet själva.

Den första spaken är den säkra kontexten, ett begrepp från W3C för en sida som levereras över HTTPS eller från den egna maskinen. Webbläsare reserverar en växande uppsättning funktioner för säkra kontexter. Service workers är en av dem: `navigator.serviceWorker` är deklarerad bara för säkra kontexter, så på en sida som laddats över okrypterad HTTP finns egenskapen helt enkelt inte, och inte heller någon funktion som bygger på den, med offlinecachning först i ledet. Specifikationen för säkra kontexter rekommenderar också att den som skriver nya specifikationer skyddar känsliga funktioner på samma sätt, så begränsningen är förvalet för nytt arbete och inget undantag som prövas från fall till fall.

Den andra spaken är blandat innehåll. När en sida själv är HTTPS sorterar webbläsaren var och en av dess hämtningar enligt W3C:s regler för Mixed Content. Bilder, ljud och video som begärs över `http://` uppgraderas automatiskt till `https://`, och de misslyckas om den adressen inte svarar. Skript, stilmallar och dataförfrågningar blockeras helt med ett nätverksfel. En enda kvarglömd okrypterad adress i en mall kan därför slå ut en hel funktion och inte bara en bild.

Det lämnar mycket lite åt port 80. En lyssnare som besvarar varje begäran med en permanent omdirigering till samma sökväg på `https://` gör fortfarande nytta, eftersom folk skriver bara namnet och gamla länkar fortfarande säger `http://`. Certifikatutfärdarens HTTP-utmaning kommer också dit när ett

certifikat utfärdas eller förnyas. HSTS låter sedan en återkommande webbläsare hoppa över även omdirigeringen. Allt utöver de uppgifterna på port 80 är en sida som serveras utan garantierna ovan.

Loopback är det enda bestående undantaget för vanliga webbplatser. Specifikationen räknar `localhost`, `127.0.0.1` och `::1` som potentiellt pålitliga, så en sida som serveras från utvecklarens egen maskin får funktionerna för säkra kontexter utan något certifikat. Den bekvämligheten för utvecklare är upphovet till ett av felen längre ned.

Så ser det ut i praktiken

Företagets egen webbplats följer den regeln. På produktionsvärden besvarar Caddy varje webbförfrågan och förnyar HTTPS-certifikatet själv, från Let's Encrypt, så att hålla certifikatet aktuellt är webbserverns jobb och inte en människas. Samma sidor publiceras också som en onion-tjänst i Tor, och den adressen börjar med `http://` med avsikt. En onion-adress kodar in tjänstens publika identitetsnyckel. Tor autentiserar tjänsten och krypterar anslutningen innan HTTP utväxlas, vilket är skälet till att den här spegeln använder HTTP över Tor. Samma sida länkar till Qualys SSL Labs, som betygsätter den aktiva HTTPS-konfigurationen åt vem som helst som frågar.

Kunduppdragen visar regeln om säkra kontexter från andra hållet. För en plattform vars användare ofta är sjöfolk och fältpersonal på mobilen, och ofta offline, levererade företaget fullständigt stöd för Progressive Web App, med en service worker som cachar typsnitt, bilder, ljud, video och CDN-resurser enligt CacheFirst. Eftersom service workers egentligen bara fungerar som de ska över HTTPS testades beteendet offline och vid installation över HTTPS på riktiga enheter i stället för att antas. Samma behov syns i verktygen för ett monorepo på flera språk med 578 mål i go-task, där det lager som företaget skrev lägger en HTTPS-variant för att testa PWA och mobil bredvid utvecklingslägena native och Docker.

Var det går fel

Skripten slutar köras efter bytet till HTTPS, och en del bilder försvinner. Sidorna innehåller fortfarande `http://`-adresser, hårdkodade i en mall, sparade i innehållet i ett CMS eller inklistrade med ett kodstycke från tredje part. Webbläsaren blockerar skripten och stilmallarna och uppgraderar bilderna, som sedan går sönder överallt där den andra värden saknar HTTPS. Webbläsarens

konsol namnger varje blockerad adress. Sök igenom mallarna och det sparade innehållet efter det gamla schemat, och skriv om varje träff som `https://` eller som en sökväg relativ till webbplatsen:

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

Ett huvudfält `Content-Security-Policy: upgrade-insecure-requests` bär sidorna genom uppstädningen, men det lämnar de sparade adresserna som de var.

Webbplatsen omdirigerar till sig själv tills webbläsaren ger upp. Ett CDN eller en lastbalanserare avslutar TLS och skickar begäran vidare till applikationen över okrypterad HTTP. Applikationen ser `http`, omdirigerar till `https`, proxyn skickar det nya försöket vidare som `http` igen, och slingan slutar med ett fel om för många omdirigeringar. Samma blindhet får applikationen att bygga absoluta länkar som börjar med `http://`. Proxyn måste föra det ursprungliga schemat vidare, i standardhuvudet `Forwarded: proto=https` från RFC 7239 eller i det äldre `X-Forwarded-Proto`, och applikationen ska bara tro på huvudet när det kommer från proxyns egen adress, eftersom vilken klient som helst kan skicka det. Låt sedan ett enda lager äga omdirigeringen, normalt det som håller certifikatet.

En sessionskaka som sattes över HTTPS färdas i klartext. RFC 9110 varnar för att kakor inte respekterar den gräns som specifikationen drar mellan de två ursprungen. Enligt reglerna för kakor i RFC 6265 skickas en kaka utan attributet `Secure` även i okrypterade `http://`-begäranden till samma värd. Det räcker att en besökare skriver bara namnet för att den ska skickas, och detsamma gäller en angripare i nätverket som besvarar en helt orelaterad okrypterad begäran med en omdirigering till webbplatsens `http://`-adress. Sätt `Secure` på varje kaka som webbplatsen utfärdar, och lägg till HSTS så att webbläsaren aldrig gör den okrypterade begäran till att börja med.

Service workern körs på den bärbara datorn men saknas på telefonen. Utvecklaren öppnar `http://localhost:3000`, en säker kontext, och allt fungerar. Telefonen i samma nätverk öppnar `http://192.168.1.20:3000`, som inte är en, så `navigator.serviceWorker` är odefinierad, ett bibliotek som letar efter den hoppar över registreringen, och offlineläget uteblir i det tysta. Servera utvecklingsbygget över HTTPS det också. Caddy, för att ta ett exempel, utfärdar certifikat för lokala namn och IP-adresser från sin egen lokala certifikatutfärdare

och försöker lägga till den roten i maskinens förtroendelager. Dokumentationen varnar för att steget inte är garanterat och att det är administratörens jobb att få in roten i varje förtroendelager, och en testtelefon måste få den roten för hand.

Läs mer

- RFC 9110, section 4.2.2: https URI Scheme — schemat, dess standardport och regeln att en klient måste kontrollera serverns identitet.
- Secure Contexts — vilka sidor som räknas som säkra, varför loopback räknas dit och rådet att begränsa nya funktioner.
- Mixed Content — vilka begäranden från en HTTPS-sida som uppgraderas och vilka som blockeras.
- RFC 7239: Forwarded HTTP Extension — standardhuvudet som en proxy använder för att föra vidare det schema som besökaren använde.
- Automatic HTTPS — Caddys egen redogörelse för dess certifikat, dess omdirigering från HTTP och dess lokala certifikatutfärdare.

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 HTTPS?” 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/https/