**Universitatea Tehnică a Moldovei**

**Lucrare de laborator 3**

**LA DISCIPLINA:** PROGRAMAREA IN RETEA

**CU TEMA:** HTTP Client

**Student:**

Lupei Nicolae FI-171

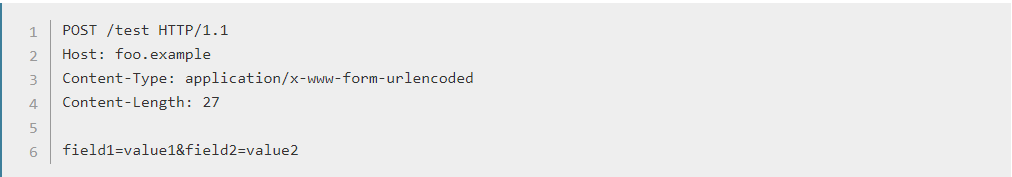
**Profesor:**

Boldumac Oleg

Chișinău – 2020

**Întrebări la apărarea laboratorului:**

1. **Cum este formatat corpul unei cereri HTTP pentru o cerere HTTP de tip POST ?**





1. **De unde știe un client HTTP ce tip de conținut trimite serverul HTTP ?**

**Content-Type** - antet entitate este utilizată pentru a indica tipul de suport al resursei.

Content-Type antet spune clientului care este de fapt tipul de conținut returnat. Navigatoarele vor face MIME adulmecând în unele cazuri și nu vor urmări neapărat valoarea acestui antet; pentru a preveni acest comportament, antetul X-Content-Type-Options poate fi setat pe nosniff.

1. **Cum decide un client dacă ar trebui să aibă încredere în certificatul unui server ?**

Când clientul verifică un certificat, există trei posibilități:

* Certificatul este semnat de un CA în care clientul are deja încredere (și pentru care cunoaște cheia publică). În acest caz, clientul tratează certificatul ca fiind valid.
* Certificatul este semnat de un CA despre care clientul nu are deloc cunoștințe. În acest caz, clientul tratează certificatul ca fiind nevalid (iar browserul va afișa probabil un mesaj de avertizare în loc să încarce pagina).
* Certificatul este semnat de un CA pe care clientul nu îl cunoaște, dar care are un certificat semnat de o CA pe care clientul îl cunoaște. (În acest caz, serverul trebuie să trimită, de obicei, atât propriul său certificat, cât și certificatul CA - denumit „CA intermediar” - care a semnat certificatul său). Întrucât certificatul de CA intermediar este semnat de o CA în care clientul are deja încredere, știe că poate avea încredere în el și, din moment ce certificatul serverului este semnat de CA intermediar, clientul știe că poate avea încredere și în el.

1. **Care este problema principală cu certificatele autosemnate ?**

Riscurile sunt pentru client. Obiectivul certificatului de server SSL este că acesta este utilizat de client pentru a cunoaște cheia publică a serverului, cu un anumit nivel de garanție că tasta aparține într-adevăr serverului dorit. Garanția provine de la CA: CA trebuie să efectueze verificări extinse ale identității solicitantului înainte de eliberarea certificatului.

Când un client (utilizatorul și browserul său Web) „acceptă” un certificat care nu a fost emis de unul dintre CA în care clientul are încredere (CA, care au fost încorporate în Windows de Microsoft), atunci riscul este că clientul este vorbesc în prezent cu un server fals, adică este atacat. Rețineți că atacurile pasive (atacatorul observă datele, dar nu le modifică în niciun fel) sunt zădărnicite de SSL, indiferent dacă certificatul CA a fost emis sau nu de un CA principal.

În general, nu doriți să vă instruiți utilizatorii să ignore avertismentul înfricoșător de securitate din browser, deoarece acest lucru îi face vulnerabili la astfel de atacuri de răspundere a serverului (care nu sunt atât de greu de montat, de exemplu cu intoxicații DNS ). Pe de altă parte, dacă puteți confirma, printr-un alt mod, că certificatul este autentic că o dată , atunci browserul va aminti certificatul și nu va afișa avertismente pentru vizitele ulterioare, atâta timp cât va fi utilizat același certificat auto-semnat . Convergența recent propusăPKI este o extensie a acestui principiu. Rețineți că acest „certificat amintit” păstrează atâta timp cât certificatul este neschimbat, deci doriți cu adevărat să setați data de expirare a certificatului dvs. semnat în viitorul îndepărtat (dar nu peste 2038 dacă doriți să evitați problemele de interoperabilitate ).

Trebuie menționat că, din moment ce un certificat semnat cu sine nu este „administrat” de către o CA, nu există nicio revocare posibilă. Dacă un atacator îți fură cheia privată, pierzi definitiv , în timp ce certificatele emise de CA au încă o teoretică netă de siguranță a revocării (o modalitate prin care CA poate declara că un anumit certificat este putred). În practică, browserul Web actual nu verifică starea revocării.

1. **Conexiunea persistentă HTTP – care sunt principalele beneficii ?**

O conexiune persistentă (conexiune persistentă HTTP ) este un canal de comunicare de rețea care rămâne deschis pentru alte cereri și răspunsuri HTTP în loc să se închidă după un singur schimb.

HTTP are o funcție de conexiune persistentă care permite canalului să rămână deschis decât să fie închis după un schimb de date solicitat. TCP este un protocol orientat către conexiune: începe o conexiune după confirmarea de la ambele capete că sunt disponibile și deschise la un schimb de date. Într-o conexiune non-persistentă, canalul se închide atunci când o gazdă semnalează că vrea să încheie comunicările sau când a trecut o anumită perioadă de timp fără schimb de date. Pentru a menține o conexiune persistentă, pachetele TCP mențin în viață sunt trimise pentru a preveni terminarea conexiunii.

O conexiune deschisă este mai rapidă pentru schimburi frecvente de date. Comunicările generale sunt salvate lăsând o conexiune deschisă, mai degrabă decât deschiderea și închiderea sesiunilor pentru fiecare solicitare. Conexiunile persistente pot fi de asemenea utilizate cu API-uri pentru a permite serverelor să împingă date către clienți. Alte avantaje ale conexiunilor persistente includ congestia redusă , latența și utilizarea procesorului și memoria datorită numărului mai mic de conexiuni; erorile pot fi, de asemenea, raportate fără a închide conexiunea.

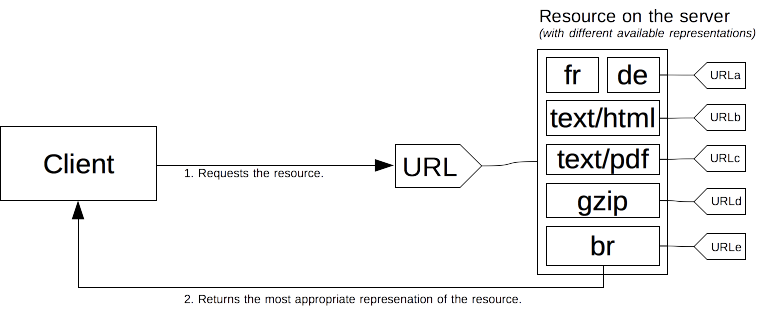
Conexiunile persistente adăugate pentru HTTP 1.0 au folosit un antet suplimentar pentru a solicita clientului să mențină conexiunea în viață; HTTP 1.1 presupune ca toate conexiunile să fie persistente, dacă nu se specifică altfel. HTTP 2.0 extinde conexiunea persistentă pentru a permite schimbarea simultană a numeroase cereri și returnări de date pe o singură conexiune.

Conexiunile persistente sunt, de asemenea, numite HTTP menținerea în viață sau reutilizarea conexiunii HTTP.

1. **Ce este negocierea conținutului în HTTP și cum are loc ?**

În HTTP , negocierea conținutului este mecanismul utilizat pentru difuzarea diferitelor reprezentări ale unei resurse la același URI, astfel încât agentul utilizator poate specifica care este cel mai potrivit pentru utilizator (de exemplu, ce limbă a unui document, ce format de imagine sau care conține codificarea).

Un document specific se numește *resursă* . Când un client dorește să-l obțină, îl solicită utilizând adresa URL. Serverul utilizează această adresă URL pentru a alege una dintre variantele pe care le furnizează - fiecare variantă fiind numită *reprezentare* - și returnează această reprezentare specifică clientului. Resursa globală, precum și fiecare dintre reprezentări, au o adresă URL specifică. Modul în care se alege o reprezentare specifică la apelul resursei este determinat de *negocierea conținutului* și există mai multe modalități de negociere între client și server.



Determinarea celei mai potrivite reprezentări se face printr-unul din cele două mecanisme:

* [Anteturi HTTP](https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers) specifice de către client ( *negociere bazată pe server* sau *negociere proactivă* ), care este modul standard de negociere a unui anumit tip de resursă.
* Cele [300](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/300)(Alegeri multiple) sau [406](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status/406)(Neacceptabilă) [HTTP coduri de răspuns](https://developer.mozilla.org/en-US/docs/Web/HTTP/Status) de către serverul ( *negociere-condus agent* sau *negociere reactivă* ), care sunt utilizate ca mecanisme alternative.

De-a lungul anilor , au fost propuse alte propuneri de negociere de conținut, cum ar fi *negocierea transparentă de conținut* și Alternatesantetul. Nu au reușit să tragă și au fost abandonate.

1. **Care sunt tipurile de negociere a conținutului HTTP ?**

**Negociere de conținut bazată pe server**

În negocierea conținutului bazată pe server sau în negocierea conținutului proactiv, browserul (sau orice alt tip de user-agent) trimite mai multe anteturi HTTP împreună cu adresa URL. Aceste anteturi descriu alegerea preferată a utilizatorului. Serverul le folosește ca indicii, iar un algoritm intern alege cel mai bun conținut pentru a-l servi clientului. Algoritmul este specific serverului și nu este definit în standard. Vezi, de exemplu, algoritmul de negociere Apache .

**Negocierea axată de agenți**

Negocierea bazată pe server suferă de câteva dezavantaje: nu se dezvoltă bine. Există un antet pe fiecare caracteristică utilizat în negociere. Dacă doriți să utilizați dimensiunea ecranului, rezoluția sau alte dimensiuni, trebuie creat un nou antet HTTP. Trimiterea anteturilor trebuie făcută la fiecare cerere. Acest lucru nu este prea problematic cu puține anteturi, dar cu eventualele înmulțiri ale acestora, dimensiunea mesajului ar duce la o scădere a performanței. Cu cât sunt trimise mai multe antete mai precise, cu atât mai multă entropie este trimisă, permițând mai multe amprente HTTP și îngrijorări de confidențialitate corespunzătoare.

De la începuturile HTTP, protocolul a permis un alt tip de negociere: negociere bazată pe agent sau negociere reactivă . În această negociere, atunci când se confruntă cu o solicitare ambiguă, serverul trimite înapoi o pagină care conține legături către resursele alternative disponibile. Utilizatorului i se prezintă resursele și îl alege pe cel pe care să-l folosească.

1. **Ce este un ETag în HTTP și cum funcționează ?**

ETag sau tag - ul entitate face parte din HTTP , protocolul pentru World Wide Web . Este unul dintre mai multe mecanisme pe care le oferă HTTP pentru validarea cache-ului Web , care permite unui client să facă cereri condiționate. Acest lucru permite cașelor să fie mai eficiente și economisește lățimea de bandă, deoarece un server Web nu trebuie să trimită un răspuns complet dacă conținutul nu s-a schimbat. ETag-urile pot fi, de asemenea, utilizate pentru controlul optim al concurentei , [1] ca o modalitate de a ajuta la prevenirea actualizărilor simultane ale unei resurse de a se suprascrie reciproc.

Un ETag este un identificator opac atribuit de un server Web unei versiuni specifice a unei resurse găsite la o adresă URL . Dacă reprezentarea resurselor de la adresa URL se schimbă vreodată, se atribuie un ETag nou și diferit. Folosite în acest mod, ETag-urile sunt similare cu amprentele și pot fi comparate rapid pentru a determina dacă două reprezentări ale unei resurse sunt aceleași.

1. **Diferența dintre protocoalele fără stare și cele cu stare. Cărui tip îi aparține HTTP ?**

| **STATELESS PROTOCOL** | **STATEFUL PROTOCOL** |
| --- | --- |
| Protocolul fără stat nu necesită ca serverul să rețină informațiile despre server sau detaliile sesiunii. | Protocolul de stat necesită serverul să salveze informațiile despre starea și sesiunea. |
| În Protocolul fără stat, nu există o dependență strânsă între server și client. | În protocolul de stat, există o dependență strânsă între server și client |
| Proiectul protocolului fără stat simplifică designul serverului. | Proiectarea protocolului Stateful face ca designul serverului să fie foarte complex și greu. |
| Protocoalele fără stat funcționează mai bine la momentul prăbușirii, deoarece nu există nicio stare care trebuie să fie restaurată, un server eșuat poate pur și simplu reporni după un crash. | Protocolul de stat nu funcționează mai bine în momentul prăbușirii, deoarece serverul statistic trebuie să păstreze informațiile despre starea și detaliile sesiunii din stările interne. |
| Protocoalele apatride gestionează tranzacția foarte rapid. | Protocoalele statale gestionează tranzacția foarte lent. |
| Protocoalele apatride sunt ușor de implementat în Internet. | Protocoalele statale sunt logic greu de implementat în Internet. |

1. **Avantajele cheie ale HTTP/2 în comparație cu HTTP/1.1**

**Beneficii:**

* Prelucrare scăzută în analizarea datelor - o propunere de valoare critică în HTTP / 2 vs HTTP1.
* Mai puțin predispus la erori.
* Amprenta de rețea mai ușoară.
* Utilizarea eficientă a resurselor de rețea.
* Eliminarea problemelor de securitate asociate cu natura textuală a HTTP1.x, cum ar fi atacurile de împărțire a răspunsurilor.
* Permite alte funcții ale HTTP / 2 incluzând compresia, multiplexarea, prioritizarea, controlul fluxului și manipularea eficientă a TLS.
* Reprezentarea compactă a comenzilor pentru o procesare și implementare mai ușoară.
* Eficient și robust în ceea ce privește procesarea datelor între client și server.
* Reducerea latenței rețelei și un randament îmbunătățit.

1. **Ce este un tip MIME, din ce constă și pentru ce se folosește ?**

Un tip media (cunoscut și sub denumirea de Extensii multifuncționale Internet Mail sau tip MIME ) este un standard care indică natura și formatul unui document, fișier sau sortiment de octeți. Este definit și standardizat în RFC 6838 al IETF .

Cele Internet Assigned Numbers Authority (IANA) este responsabil pentru toate tipurile MIME oficiale, și puteți găsi cele mai actualizate și complete la lista lor Media Tipuri de pagini.

Cel mai simplu tip MIME constă dintr-un tip și un subtip ; acestea sunt fiecare șiruri care, atunci când sunt concatenate cu o bară /între ele, conțin un tip MIME. Nu este permis nici un spațiu alb într-un tip MIME:

1. **Care este diferența dintre GET și POST ?**

**GET** solicită o reprezentare a resursei specificate. Rețineți că GET nu trebuie utilizat pentru operațiuni care provoacă efecte secundare, cum ar fi utilizarea acestuia pentru acțiuni în aplicații web. Un motiv pentru acest lucru este că GET poate fi utilizat în mod arbitrar de către roboți sau crawler, care nu ar trebui să ia în considerare efectele secundare pe care le poate provoca o solicitare.

**POST** trimite datele care urmează să fie procesate (de exemplu, dintr-un formular HTML) la resursa identificată. Datele sunt incluse în corpul cererii. Aceasta poate duce la crearea unei noi resurse sau la actualizări ale resurselor existente sau ale ambelor.

1. **Care este diferența dintre PUT și POST ?**

**PUT** pune un fișier sau resursă la un anumit URI, și exact la acel URI. Dacă există deja un fișier sau resursă la acel URI, PUT înlocuiește acel fișier sau resursă. Dacă nu există un fișier sau resursă acolo, PUT creează unul. PUT este idempotent , dar paradoxal răspunsurile PUT nu sunt memorate în cache.

**POST** trimite date către un anumit URI și se așteaptă ca resursa la acel URI să se ocupe de solicitare. Serverul web în acest moment poate determina ce să facă cu datele din contextul resursei specificate. Metoda POST nu este idempotentă , însă răspunsurile POST sunt memorate în cache atâta timp cât serverul stabilește anteturile corespunzătoare Cache-Control și expiră.

Diferența fundamentală între solicitările POST și PUT se reflectă în sensul diferit al URI-ului Cerere. URI dintr-o solicitare POST identifică resursa care va gestiona entitatea închisă. Resursa ar putea fi un proces de acceptare a datelor, o poartă de acces la un alt protocol sau o entitate separată care acceptă adnotări. În schimb, URI într-o solicitare PUT identifică entitatea anexată cererii - agentul utilizator știe ce URI este destinat, iar serverul NU TREBUIE să încerce să aplice cererea la alte resurse. Dacă serverul dorește ca cererea să fie aplicată unui alt URI, TREBUIE să trimită un răspuns 301 (Mutat permanent); agentul utilizator poate MAI lua decizia sa privind redirecționarea sau nu a cererii.

1. **Care sunt metodele idempotente în HTTP și care sunt scopul lor.**

Idempotency este o proprietate a metodelor HTTP.

O metodă de solicitare este considerată idempotentă dacă efectul prevăzut pe server de mai multe cereri identice cu acea metodă este același cu efectul pentru o singură cerere. Și merită menționat că idempotența este despre efectul produs asupra stării resursei pe server și nu despre codul de stare de răspuns primit de client.

Pentru a ilustra acest lucru, luați în considerare DELETEmetoda, care este definită drept idempotentă. Acum, luați în considerare un client efectuează o DELETEsolicitare pentru a șterge o resursă de pe server. Serverul procesează solicitarea, resursa este ștersă și serverul revine 204. Apoi clientul repetă aceeași DELETE solicitare și, întrucât resursa a fost deja ștersă, serverul revine 404.

În ciuda codului de stare diferit primit de client, efectul produs de o singură DELETE solicitare este același efect al DELETE cererilor multiple către același URI.

În cele din urmă, cererile cu metode idempotente pot fi repetate automat dacă apare o defecțiune de comunicare înainte ca clientul să poată citi răspunsul serverului. Clientul știe că repetarea cererii va avea același efect prevăzut , chiar dacă cererea inițială a reușit, deși răspunsul ar putea fi diferit.

1. **Cum sunt identificate resursele în protocolul HTTP ?**

Ținta unei solicitări HTTP se numește „resursă”, a cărei natură nu este definită în continuare; poate fi un document, o fotografie sau orice altceva. Fiecare resursă este identificată de către un URI (Uniform Resource Identifier ) utilizat în HTTP pentru identificarea resurselor.

Identitatea și locația resurselor de pe Web sunt date în mare parte de o singură adresă URL (Uniform Resource Locator, un fel de URI). Există, uneori, motive pentru care identitatea și locația nu sunt date de același URI: HTTP folosește un antet HTTP specific, Alt-Svcatunci când resursa solicitată dorește ca clientul să îl poată accesa într-o altă locație.

1. **Care sunt metodele sigure și nesigure în HTTP ?**

Ce este metode sigure în HTTP

Este vorba de metode HTTP care nu schimbă resursa pe server. De exemplu, utilizarea unui GET sau o cerere HEAD pe o adresă URL a resursei nu ar trebui să schimbe NICIODATĂ resursa. Metodele sigure pot fi memorate în cache și prefațate fără repercusiuni sau efecte secundare asupra resursei. Iată un exemplu de metodă sigură

GET / comanda / 123 HTTP / 1.1

Aceasta va prelua comanda cu orderId 123. Indiferent de câte ori executați această metodă, comanda din server nu va fi modificată sau afectată. De aceea, metoda GET este o metodă sigură.

Care sunt metodele Idempotent în HTTP

Acestea sunt metode care sunt ferite de mai multe apeluri, adică produc același rezultat indiferent de câte ori le sunați. Ei schimbă resursa în Server de fiecare dată când le sunați, dar rezultatul final este întotdeauna același. Matematica este un loc bun pentru a explica metodele idempotente, ia în considerare următorul exemplu:

int i = 30 ; // idempotent

i + + ; // nu este idempotent

Aici operația de atribuire este idempotente, indiferent de câte ori executați această afirmație, eu va fi întotdeauna 4. doilea exemplu nu este idempotente . Executarea acestui lucru de 10 ori va avea ca rezultat un rezultat diferit de cel al rulării de 5 ori. Deoarece ambele exemple se schimba valoarea i , ambele sunt metode non-sigure.

Idempotența este un lucru important în timp ce construiți o API RESTful tolerantă la erori. De asemenea, idempotența este motivul pentru care ar trebui să utilizați PUT over POST pentru a actualiza o resursă în REST .

De exemplu, să presupunem că un client dorește să actualizeze o resursă prin POST. Deoarece POST nu este o metodă idempotentă , apelarea acesteia de mai multe ori poate duce la actualizări incorecte.

În lumea reală, este foarte liniștit că cererea dvs. POST se poate termina, ceea ce se va întâmpla cu resursa care. Resursa este actualizată de fapt? S-a întâmplat expirarea în timpul trimiterii cererii către server sau răspunsul către client?

Putem încerca din nou în siguranță sau trebuie să ne dăm seama mai întâi de ce s-a întâmplat cu resursa? Folosind metode idempotente precum PUT, nu trebuie să răspundeți la această întrebare, dar putem retrimite în siguranță cererea până când nu vom primi în realitate un răspuns din server.

Puteți citi, de asemenea, RESTful Web Services de Leonard Richardson, Sam Ruby și David Heinemeier Hansson pentru mai multe detalii despre maparea HTTP și REST.

Iată o imagine de ansamblu a metodelor HTTP sigure și idempotente:

GET este atât sigur cât și idempotent.

HEAD este, de asemenea, sigur și idempotent.

OPTIONS este, de asemenea, sigur și idempotent.

PUT nu este sigur, dar idempotent.

DELETE nu este sigur, dar idempotent.

POST-ul nu este nici sigur, nici idempotent.

PATCH nu este nici sigur, nici idempotent.

1. **Pentru ce este nevoie de cURL ?**

CURL, adesea doar „curl”, este un instrument gratuit pentru linia de comandă. Utilizează sintaxa URL pentru a transfera date către și de pe servere. bucla este o utilizare pe scară largă, datorită capacității sale de a fi flexibile și completează sarcini complexe. De exemplu, puteți utiliza curl pentru lucruri precum autentificarea utilizatorului, postare HTTP, conexiuni SSL, suport proxy, încărcări FTP și multe altele! De asemenea, puteți face lucruri simple cu ondularea, cum ar fi descărcarea paginilor web și a imaginilor web. Citiți mai departe pentru a afla dacă ar trebui să folosiți ondularea și, dacă da, cazurile de utilizare obișnuite care vă vor începe .

1. **Pentru ce este nevoie de HTTP Proxy?**

Practic, un server proxy acționează ca un om de mijloc între server și client care servește computere

Permite computerelor client să realizeze conexiuni de rețea indirecte la alte servicii de rețea. Dacă utilizați server proxy, computerele client se vor conecta mai întâi la serverul proxy, solicitând unele resurse precum pagini web, jocuri, videoclipuri, mp3, cărți electronice, orice alte resurse disponibile de pe diverse servere prin Internet. Imediat ce primești o astfel de solicitare, serverul proxy va căuta resursele din cache-ul din hard disk-ul local. Dacă resursele au fost memorate în cache înainte, serverul proxy le va returna calculatoarelor client. Dacă nu este memorat în cache, se va conecta la serverele relevante și va solicita resursele în numele computerelor client. Apoi „stochează” resursele de pe serverele de la distanță și returnează direct cererile ulterioare pentru același conținut.

În zilele noastre, folosim server proxy în diverse scopuri, precum partajarea conexiunilor de internet pe o rețea locală, ascundem adresa noastră IP, implementăm controlul accesului la internet, accesăm site-urile blocate etc. Mai jos sunt câteva beneficii de ce utilizează serverul proxy:

Pentru a partaja conexiunea la Internet pe LAN. Unele întreprinderi mici și familii au mai multe calculatoare, dar cu o singură conexiune la Internet, pot partaja conexiune la Internet pentru alte computere din LAN cu un server proxy.

Pentru a accelera navigarea pe Internet. Dacă utilizați server proxy, toate cererile de la computerele client vor ajunge la serverul proxy la început, dacă serverul proxy a pus în cache resursele necesare pe hard disk-ul local înainte cu funcția de cache web, clienții vor primi feedback direct de la serverul proxy, să fie mai rapid decât accesul direct.

Pentru a ascunde adresa IP a computerului client, astfel încât să poată naviga anonim, aceasta este în mare parte din motive de securitate. Un server proxy poate acționa ca un intermediar între computerul utilizatorului și internet pentru a preveni atacul și accesul neașteptat.

Pentru a implementa controlul accesului la Internet precum autentificarea conexiunii la Internet, controlul lățimii de bandă, controlul timpului online, filtrul web al internetului și filtrul de conținut etc.

Pentru a ocoli restricțiile de securitate și filtrele. De exemplu, multe birouri de lucru au blocat facebook și myspace, cu toate acestea, puteți utiliza serverul proxy pentru a ocoli aceste restricții și pentru a accesa site-urile blocate cu ușurință.

Pentru a scana conținutul de ieșire, de exemplu, pentru protecția împotriva scurgerii de date.

Pentru a evita restricțiile regionale. De exemplu, un server care utilizează geolocalizarea bazată pe IP pentru a restricționa serviciul într-o anumită țară poate fi accesat folosind un proxy localizat în țara respectivă pentru a accesa serviciul.

1. **Diferența dintre autentificare și autorizare**

Accesul la un sistem este protejat atât prin autentificare, cât și prin autorizare. Orice încercare de acces la sistem ar putea fi autentificată prin introducerea de acreditări valide, dar poate fi acceptată numai după autorizarea cu succes. Dacă încercarea este autentificată, dar nu este autorizată, sistemul va refuza accesul la sistem.

|  |  |
| --- | --- |
| **Autentificare** | **Autorizare** |
| Autentificarea confirmă identitatea dvs. pentru a acorda acces la sistem. | Autorizarea stabilește dacă sunteți autorizat să accesați resursele. |
| Este procesul de validare a acredităților utilizatorilor pentru a obține acces utilizator. | Este procesul de a verifica dacă accesul este permis sau nu. |
| Acesta stabilește dacă utilizatorul este ceea ce pretinde că este. | Stabilește ce utilizator poate și ce nu poate accesa. |
| Autentificarea necesită de obicei un nume de utilizator și o parolă. | Factorii de autentificare necesari pentru autorizare pot varia, în funcție de nivelul de securitate. |
| Autentificarea este primul pas de autorizare, astfel încât întotdeauna vine primul. | Autorizarea se face după autentificarea cu succes. |
| De exemplu, studenții unei anumite universități trebuie să se autentifice înainte de a accesa [link-ul de](http://www.differencebetween.net/technology/difference-between-link-and-hyperlink/" \o "DIFERENȚA ÎNTRE LINK ȘI HYPERLINK) studenți de pe site-ul oficial al universității. Aceasta se numește autentificare. | De exemplu, autorizarea determină exact ce informații sunt autorizați să acceseze pe site-ul web al universității după autentificarea cu succes. |

1. **Care sunt metodele de autentificare HTTP ?**
   1. Proxy authentication
   2. Access forbidden
   3. Authentication of cross-origin images
   4. Character encoding of HTTP authentication
   5. WWW-Authenticate and Proxy-Authenticate headers
   6. Authorization and Proxy-Authorization headers
   7. Authentication schemes
      1. Basic
      2. Bearer
      3. Digest
      4. HOBA
      5. Mutual
      6. AWS4-HMAC-SHA256
2. **Modalități de identificare a utilizatorilor în HTTP**

Deci, dacă cookie-urile sunt acum persona non grata, cum putem obține aceleași rezultate, dar fără a utiliza cookie-uri? Iată 5 moduri de a vă identifica utilizatorii:

1. Utilizarea IP-ului utilizatorului

Utilizarea unei adrese IP este cea mai evidentă soluție dintre toate. Este simplu și rapid, totuși este și cel mai puțin eficient. Problema principală cu utilizarea IP-ului este că marea majoritate a utilizatorilor utilizează IP-ul dinamic, ceea ce înseamnă că adresa IP a unui utilizator astăzi nu poate fi aceeași mâine. De asemenea, dacă mai mulți utilizatori se conectează la aceeași rețea, toată lumea va avea același IP și toți vor fi indistinguibili serverului.

2. LocalStorage

O nouă caracteristică a HTML5 este LocalStorage. Îl putem conceptualiza ca cookie-uri BIG care pot stoca mult mai multe date. Cu toate acestea, spre deosebire de cookie-urile LocalStorage accesează doar javascriptul (serverul nu poate crea sau citi fișierele), de aceea trebuie să stabilim o modalitate de comunicare a datelor de identificare a utilizatorului către server (de exemplu, o solicitare AJAX).

Din păcate, ștergerea LocalStorage este, de asemenea, relativ ușoară pentru orice utilizator preocupat de confidențialitatea lor și browserele în modul incognito încă fac ca utilizatorul să fie nedetectabil.

3. Amprentarea pe pânză

Această metodă, care poate părea un pic îndepărtată, este folosită astăzi pe scară largă de rețelele publicitare importante și site-urile web. Operația se bazează pe elementul canvas HTML5 care vă permite să pictați cu javascript.

Ceea ce se întâmplă este un model particular de geometrii, iar textul este ascuns într-o zonă a fiecărei pagini. Datorită particularităților browserelor, sistemelor de operare și, mai ales, a utilizatorilor de carduri grafice, imaginea rezultată pe două computere separate poate fi făcută distinctă (deși instrucțiunile pentru generarea acesteia sunt aceleași).

Această imagine este o amprentă unică pentru orice alt computer. Dacă transmitem un hash creat cu această imagine în fiecare solicitare, putem identifica cererile făcute de la același computer fără a stoca nimic pe utilizator.

4. Comportamentul utilizatorului

Metoda de mai sus are încă o problemă: poate fi detectată. Instrumentele avansate de confidențialitate (cum ar fi browserele Tor) pot detecta când pictați pe o pânză „suspectă” și vor trimite o notificare utilizatorului.

Următoarea metodă este probabil cea mai complexă dintre toate și este disponibilă doar pentru unii giganți tehnologici, precum Google, care au resurse pentru a o elimina. Cu toate acestea, dacă este implementată corect, această nouă metodologie poate detecta dacă un utilizator a accesat site-ul dvs. de pe un alt computer!

Tehnica constă în înregistrarea comportamentului utilizatorului (mișcări ale mouse-ului, accelerare, utilizarea defilării etc.) și transmiterea informațiilor către server. Comportamentul utilizatorului este foarte personal și poate fi detectat prin gruparea algoritmilor Machine Learning.

5. Utilizarea ETAG

În cele din urmă, un echilibru între simplitate și eficiență - aceasta este metoda mea preferată. În primul rând, să explicăm cum funcționează cache-ul pe serverele moderne.

Când un utilizator solicită pentru prima dată fișierul A, serverul furnizează apoi fișierul ETAG împreună cu un cod care este o semnătură a conținutului fișierului. Când utilizatorul dorește să ordoneze fișierul a doua oară, browserul (care are fișierul și ETAG-ul în cache) trimite serverului împreună cu ETAG. Dacă fișierul nu s-a schimbat, serverul care are ETAG este același care a fost trimis de client și server, în loc să trimită fișierul a doua oară, îi spune browserului să utilizeze versiunea care este în cache. Dacă fișierul s-ar fi schimbat, serverul ar trimite clientului noua versiune cu noul ETAG. Simplu, nu?

Deci, cum putem folosi acest sistem în scopurile noastre?

Comportamentul a ceea ce nu putem schimba, dar serverul este al nostru și, prin urmare, nu îndeplinim performanța stabilită (presupunând browserul dacă va fi). Pregătiți o imagine (de la 1 × 1 pentru a nu se vedea) pe pagina noastră de pornire. Serverul, atunci când vi se solicită acea imagine își modifică comportamentul după cum urmează:

Dacă solicitarea nu are ETAG (pentru că este prima dată când utilizatorii se loghează) au generat un ID de utilizator unic și trimiteți-l browserului ca și cum ar fi imaginea ETAG.

Dacă cererea are ETAG, știm că acest ETAG este cu adevărat identificatorul nostru de utilizator, astfel încât toate cererile de la acel IP pentru sesiunea curentă sunt acel utilizator. Pentru ca browserul să nu lipsească identificatorul, vă vom spune că versiunea pe care ați imaginea în cache este corectă

Principalul dezavantaj al ETAG este să fie șters dacă utilizatorul șterge memoria cache a browserului. Cu toate acestea, având în vedere simplitatea implementării, este o modalitate excelentă de a începe mai târziu și de a completa alte metode

Aceste 5 metode sunt doar o fracțiune din cele utilizate pentru identificarea fiecărei sesiuni de utilizator. În bine sau în rău, utilizarea excesivă a acestor tehnici a devenit o ramură de cercetare foarte activă pentru companiile care se bazează pe analiza datelor utilizatorilor.

1. **HTTP cookies – pentru ce se folosește ?**

Un cookie HTTP ( cookie web, cookie pentru browser) este o mică informație pe care un server o transmite browserului web al utilizatorului. Browserul îl poate stoca și trimite înapoi cu următoarea solicitare către același server. De obicei, se folosește pentru a informa dacă două solicitări provin din același browser - păstrarea unui utilizator conectat, de exemplu. Acesta amintește informații de stat pentru protocolul HTTP fără stat