Univerza v Mariboru

FERI
Fakulteta za elektrotehniko, računalništvo in informatiko

Spletne tehnologije

Spletna avtentikacija in TLS

Niko Lukač

Osnovni pojmi

- Avtentikacija: preverjanje identite (401 Unauthorized)
  - Običajno avtentiziramo z **username+password**
  - Dodatno z **RNG kodami** (SMS, Avtentikator (fizični ali App))…
- Avtorizacija: **preverimo pravice** (403 Forbidden)
- Klasično vzdrževanje avtentikacije
  - S shranjenim stanjem (**stateful**): hranimo sejo in uporabimo **piškotke**
  - Brez stanja (**stateless**): z **žetoni**, npr. JWT / OAuth / other

Seje (sessions)

- Spomnimo se tipičnega poteka seje:
  1. Uporabnik pošlje user+hash(passwd) (HASH preko kriptografske funkcije (npr. SHA256, včasih MD5) zgeneriramo pri odjemalcu)
  2. Strežnik preveri ali se **user+hash** nahaja v bazi
  3. Strežnik po uspešni avtentikaciji shrani sejo (naključna dolga številka ali hash)
     - Seja se običajno temporarno hrani v podatkovno bazo ali pomnilnik strežniškega procesa.
  4. Strežnik pošlje piškotek, kjer je znotraj hash od seje (**session ID**)
  5. Za vsak nadaljni HTTP zahtevek odjemalec pošlje piškotek
  6. Strežnik vsakič **validira sejo** iz piškotka, da preveri če je uporabnik avtenticiran
  7. Če poteče določen čas, ali se uporabnik odjavi, bo strežnik pobrisal sejo – odjemalec pa piškotek (pri brisanju velikokrat ni sinhronizacije)

Piškotki (cookies)

- Spomnimo se delovanja piškotkov:
  - Piškotek se uporablja za shranjevanje **sej** v brskalniku, personalizaciji spletne strani (**nastavitve**) in sledenju
  - Nastavi s strani strežnika preko **Set-Cookie**
  - Dodatni atributi: domena, čas trajanja (exp), pot, Secure, SameSite, **HttpOnly**
  - Vsebino piškotkov lahko dodatno **šifriramo**, vendar pridobimo druge težave:

```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: SESS_ID=9vKnWqiZvuvVsIV1zmzJQeYUgINqXYeS; Domain=example.com; Path=/
```

Piškotki (cookies)

- Spomnimo se delovanja piškotkov:
  - Piškotek se uporablja za shranjevanje sej v brskalniku, personalizaciji spletne strani (nastavitve) in sledenju
  - Nastavi s strani strežnika preko Set-Cookie
  - Dodatni atributi: domena, čas trajanja (exp), pot, Secure, SameSite, HttpOnly
  - Vsebino piškotkov lahko dodatno šifriramo, vendar pridobimo druge težave:
  - Za šifriranje **potrebujemo JS**, torej nimamo HttpOnly piškotek, kar lahko privede do **XSS** napadov

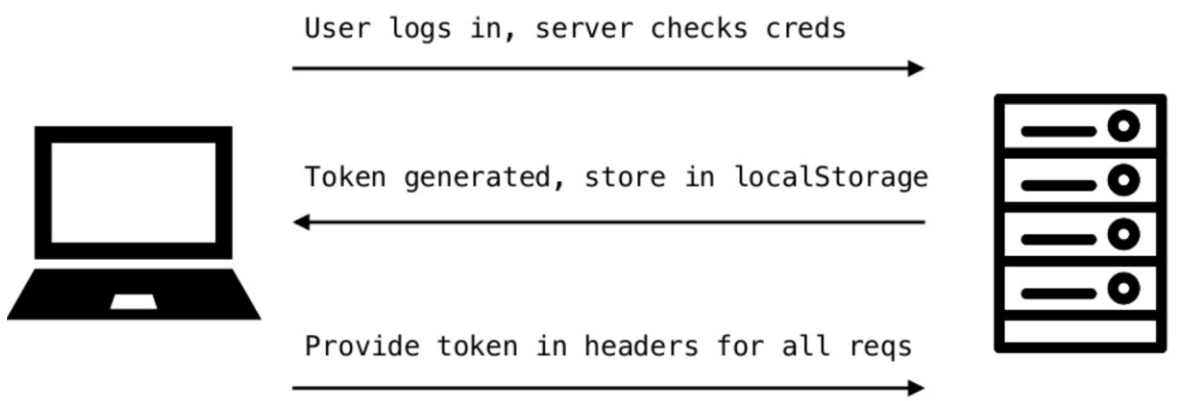
```
HTTP/1.1 200 OK
Content-type: text/html
Set-Cookie: SESS_ID=9vKnWqiZvuvVsIV1zmzJQeYUgINqXYeS; Domain=example.com; Path=/
```

Težave avtentikacije pri sejah

- **Hramba stanja**: strežnik mora hranit informacije o trenutni seji.
- **Skalabilnost**: vodenje sej preko distribuiranih sistemih ter sinhronizacija med sistemi.
- **CSRF napad**: če imamo odprto sejo (npr. preko piškotkov) lahko pride do zlorabe z zavajanjem uporabnika. Predpostavljamo, da so piškotki z zastavicami HttpOnly in Secure.

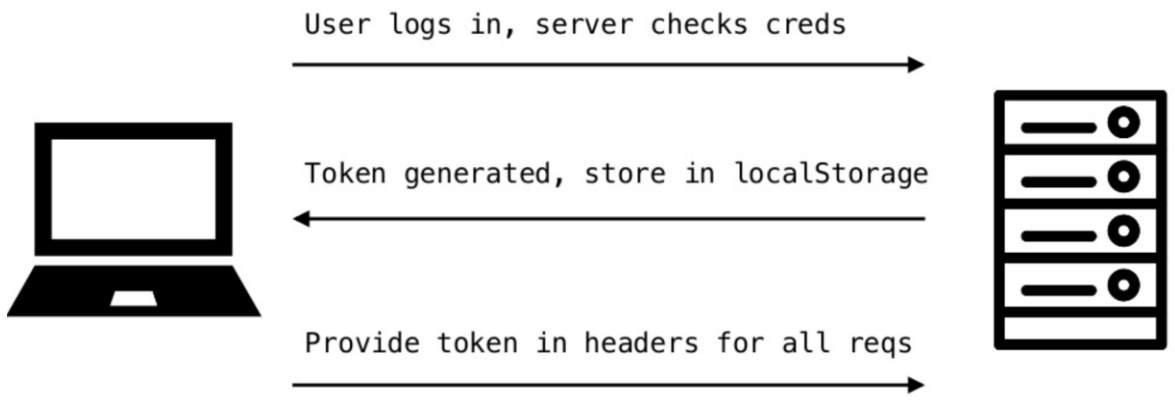
Žetoni (tokens)

- Žetoni zamenjujejo klasične seje, kar je smiselno pri REST (neodvisni oz. stateless zahtevki):
  - Uporabnik se avtenticira pri spletnem viru (**REST API**).
  - Strežnik zgenerira temporarni žeton (**običajno HASH**), podobno kot pri sejah ter ga pošlje uporabniku (brskalnik, mobilna aplikacija, itd.).
  - Uporabnik hrani žeton in ga pošlje **zraven vsakega zahtevka** (REST).
  - Dodatno se s pomočjo skritega HASH-a lahko šifrira vsebina zahtevkov.
  - Žeton se običajno hrani v pomnilniku dostopen preko **JS (LocalStorage) in ne v piškotkih**.
  - Varnostne težave?



Žetoni (tokens)

- Žetoni zamenjujejo klasične seje, kar je smiselno pri REST (neodvisni oz. stateless zahtevki):
  - Uporabnik se avtenticira pri spletnem viru (REST API).
  - Strežnik zgenerira temporarni žeton (običajno HASH), podobno kot pri sejah ter ga pošlje uporabniku (brskalnik, mobilna aplikacija, itd.).
  - Uporabnik hrani žeton in ga pošlje zraven vsakega naslednjega zahtevka (REST).
  - Dodatno se s pomočjo skritega HASH-a lahko šifrira vsebina zahtevkov.
  - Žeton se običajno hrani v pomnilniku dostopen preko JS (LocalStorage) in ne v piškotkih.
  - Varnostne težave? **napadi XSS (saj se nahajamo v JS)**



JSON Web Token (JWT)

- JWT je odprt standard (RF 7519), ki definira **kompaktno in izolirano obliko** za prenos sporočil oz. informacij kot JSON object. Kompatibilno z REST.
  - https://tools.ietf.org/html/rfc7519
- JWT omogoča **tudi digitalno podpisovanje** sporočil za doseganje avtentičnosti.
- Tipični format JWT, kjer posamezne dele kodiramo z **base64**:

header.payload.signature

- Glava JWT JSON objekta:
  - 1. Vsebuje ime algoritma s katerim se izvede podpisovanje (**HMAC SHA256** ali RSA)
  - 2. Tip objekta (JWT)

```
{
    “alg”: “H256”
    “typ”: “JWT”
}
```

JSON Web Token (JWT)

- Telo JWT JSON objekta vsebuje metapodatke (**claims**), po standardu sledeče:
  - **Registrirane**: niso obvezne vendar priporočene:
    - NPR: iss (issuer), exp (experitaion time), sub (subject), aud (audience)
  - **Javne** (public): generične informacije (npr. ime, e-mail)
  - **Privatne** (private): sporočila vezane na aplikacijo (npr. user id), ki si izmenjujejo uporabniki JWT (npr. odjemalec-strežnik)
- Primer telesa:

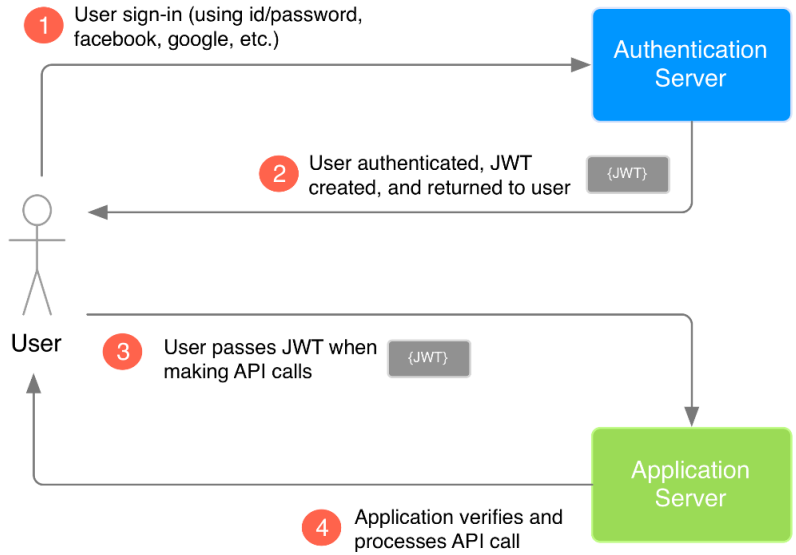
```
{
        "sub": "1234567890",
        "name": "John Doe",
        "admin": true
}
```

JSON Web Token (JWT)

- Pri JWT vsak **JSON objekt podpišemo**, kar nam enolično omogoči avtentičnost sporočila. Primer z HMAC SHA256:

HMACSHA256(base64(header)+”.”+base64(payload), secret)

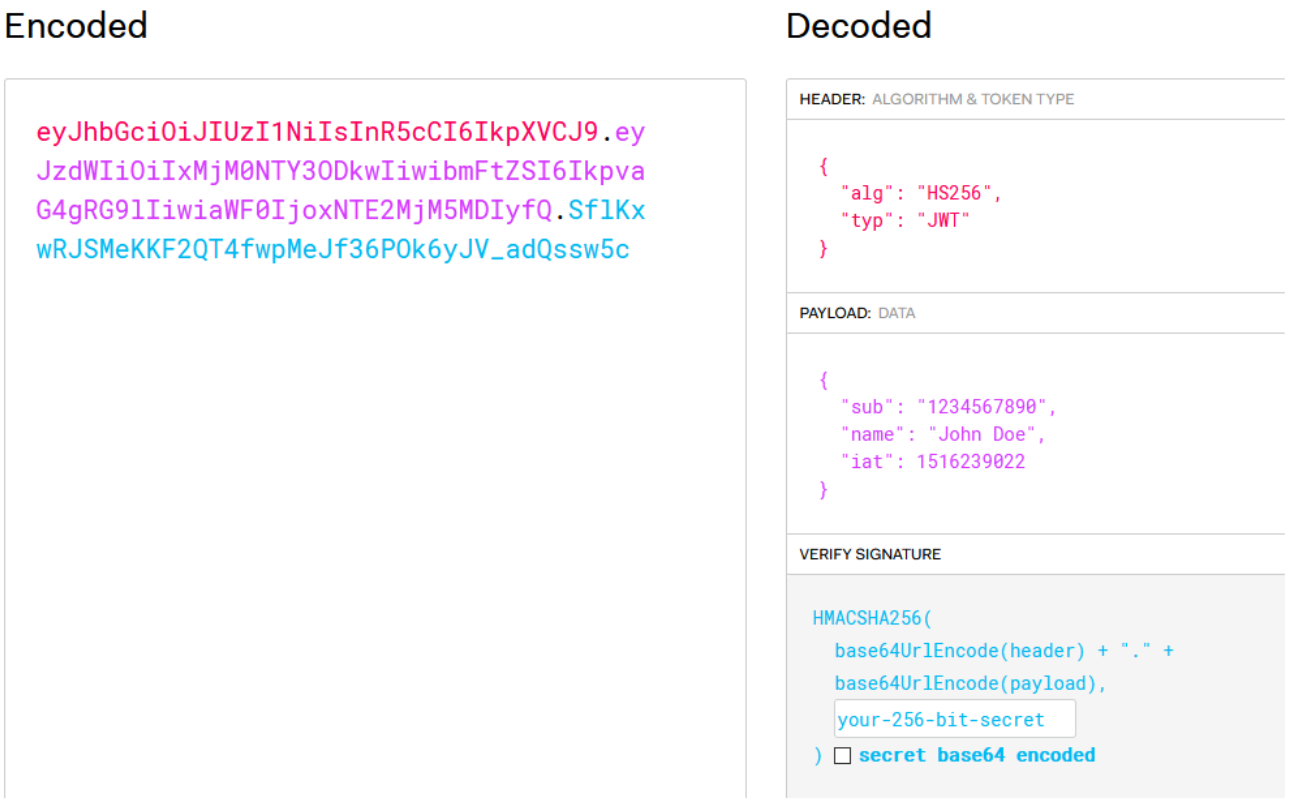
- Po avtentikaciji uporabnik uporablja JWT preko REST API:
  - Secret je skriti niz (HASH) ali API ključ, ki zamenja “sejo” in **običajno zgenerira strežnik**, potrebna izmenjava 1x po avtentikaciji
  - API ključ se lahko hrani dlje časa



Vir slike: https://www.softwaresecured.com/security-issues-jwt-authentication/

JSON Web Token (JWT)

- Primer celotnega JWT (pošiljamo preko REST) (iz https://jwt.io):



Shranjevanje JWT ?

- JWT lahko shranimo preko JS v lokalni pomnilnik kot key-value
  - localStorage je brez omejitev
  - sessionStorage se izbriše ob zaprtju spletne strani
- Prednosti
  - Shranjevanje z localStorage je vezano na protocol://host:port, ostale domene nimajo dostopa
  - Max. velikost višja od piškotkov (5 MB vs 4 KB piškotek)
- Slabosti
  - Limitirani smo na nize, potrebna serializacija
  - Pri localStorage nimamo mehanizma avtomatskega brisanja, razen v primeru nastavitev brskalnika
  - Možnost izvedbe XSS napada (JS)
  - **Imunost pred CSRF napadi**
- Praktična uporaba najboljša za javne podatke (preko API), slaba praksa za zasebne podatke

Seje vs JWT

- Prednosti sej napram JWT
  - Piškotke lahko zavarujemo z **HttpOnly**, Secure zastavicami
  - Pri piškotkih tako **nimamo možnih XSS** napadov

- Slabosti sej napram JWT
  - Strežnik mora zavarovati uporabnike pred napadi **CSRF** (lažje kot XSS)
  - Vezani smo le na **osnovne HTTP zahtevke**
  - Višja **možnost MITM** (man-in-the-middle) napada

- **Slabosti obeh**
  - Strežnik mora **vodit ID seje ali skritega niza JWT žetona** (v primeru vzpostavitve seje)
  - Pri **skalabilnosti** se to vodenje mora propagirat preko več strežnikov

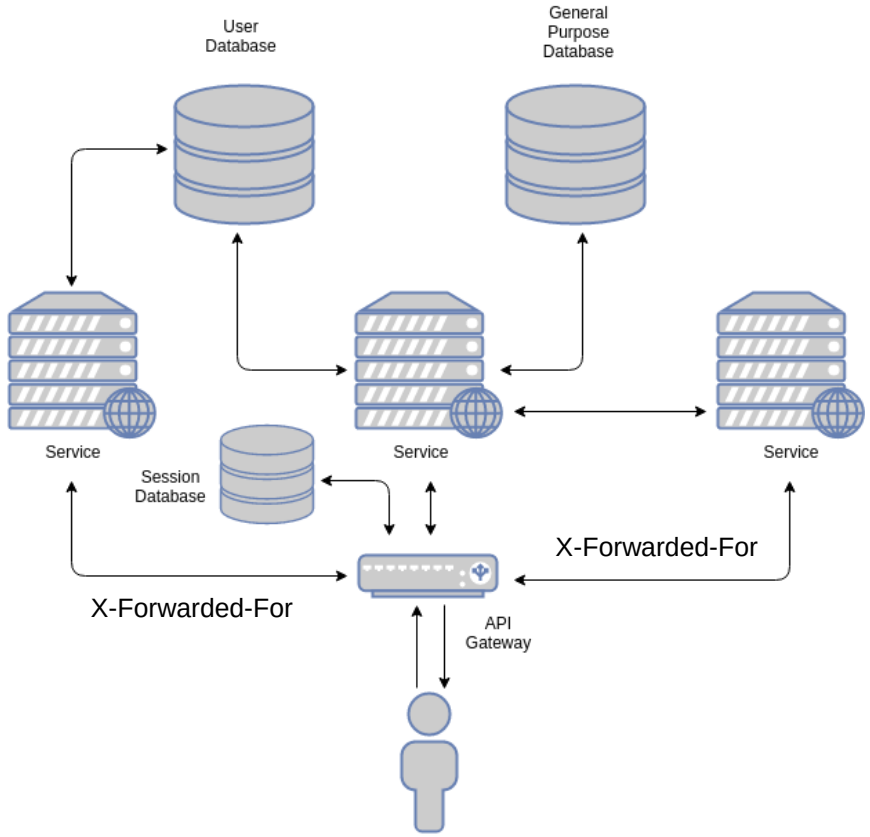
Stateless JWT

- Stateless (brez seje)
  - Na strežniku ne hranimo informacij o žetonu (skriti niz v JWT)
  - Vsak JWT zahtevek hrani informacije o avtentikaciji
    - Npr. izmenjava skritega niza odpade, zgenerira se direktno iz gesla uporabnika

- Dodatna zaščita na strežnikih
  - **Preverjanje IP**, **user-agent, 2-faktorska avtentikacija** (za generiranje skritega niza), itd.

API prehod (gateway)

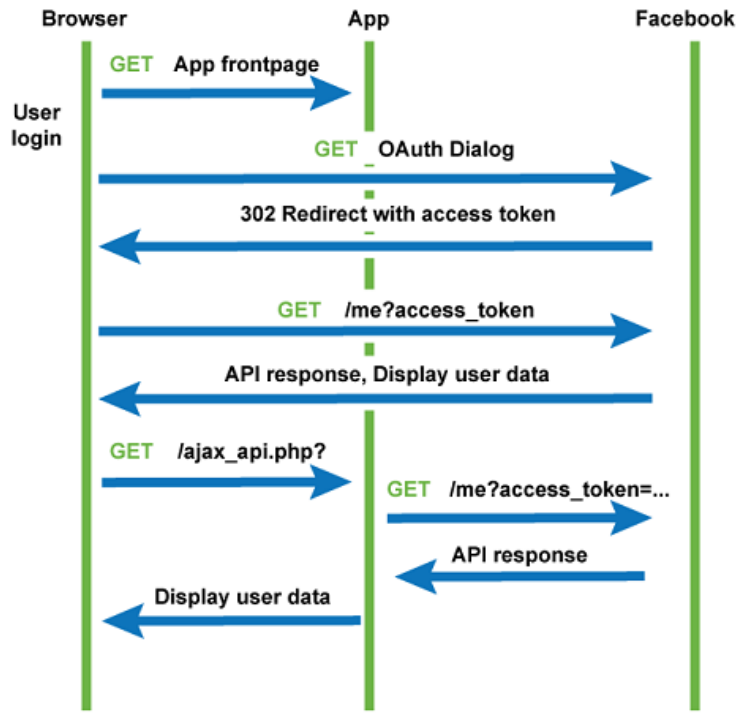
- Lahko uporabimo za seje ali za JWT
- Za klice API je JWT bolj primeren, saj so lahko REST zahtevki neodvisni
- Glavna slabost ?
  - SPOF (single point of failure)
  - Sicer enako velja za vse load balancerje…
- Skalabilnost ? Uporabljamo več API gateway-ov (npr. za vsak servis posebej, delijo si lahko enako bazo za shranjevanje API ključev -> možna skalabilnost z memcached)
- Kako **strežniki storitev zvedo IP odjemalca**?
  - "**X-Forwarded-For**: IP1, PROXY-IP-1,…, PROXY-IP-N" bo API gateway hranil v vsako **glavo HTTP zahtevka**



Vir slike: https://auth0.com/blog/stateless-auth-for-stateful-minds/

Oauth

- Odprt standard (v 2006 razvil Twitter & Google), da se uporabniki avtorizirajo preko 3rd-party strani (npr. **Facebook account, Google account itd**)
  - **Preko avtorizacije dobimo ti. psevdo-avtentikacijo**
- Popularna alternativa: **OpenID**
  - Oauth je vezan na avtorizacijo, OpenID na **avtentikacijo**, sta **komplementarna**
- Primer URL **zahtevka avtorizacije**:
  https://server.com/auth?response_type=code&client_id=CLIENT_ID&redirect_uri=REDIRECT_URI&scope=photos&state=1234zy
  - **respone-type** = potrebujemo avtorizacijsko kodo
  - **client_id** = identifikator aplikacije (registrirano pri Oauth strežniku)
  - **redirect_uri** =kam se uporabnika preusmeri po avtorizaciji z avtorizacijsko kodo
  - **scope** = pravice uporabnika, ki jih želimo
  - **state** = random niz za verifikacijo
- **Aplikacija** po avtorizaciji dobi avtorizacijsko kodo, ki pošlje na **Oauth strežnik**, od katerega nato **dobi žeton za komunikacijo**.



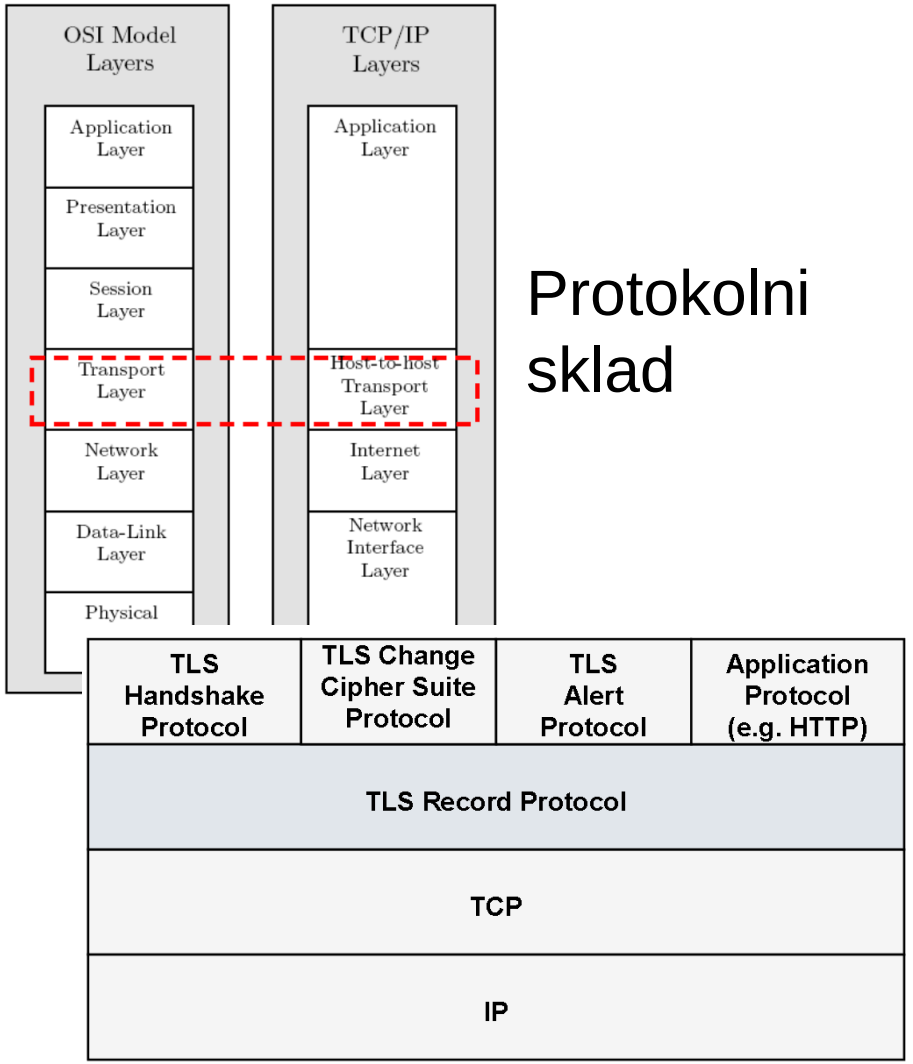
Vir slike: https://www.soapui.org/docs/oauth2/oauth2-overview/

HTTPS (SSL in TLS) – kratka zgodovina

- 1994: Netscape Communications je razvil protocol **Secure Sockets Layer, SSLv2**.
  - Ogromno ranljivosti, ni več v uporabi (deprecated) od 2011
- 1995: Netscape izda **novejšo različico SSLv3**.
  - Ogromno ranljivosti, ni več v uporabi (deprecated) od 2015
- 1999, RFC 2246 je izšel s strani IETF, **zaživel Transport Layer Security Protocol**:
  - TLS 1.0 podoben vendar nekompatibilen z SSLv3
  - Sledijo **TLS 1.1 (2006) and TLS 1.2 (2008), TLS 1.3 (2018+) danes uporabljamo**

Transport Layer Security Protocol (TLS)

- TLS je protokol kriptografskih storitev, ki temelji na infrastrukturi **PKI (public key infrastructure).**
- Vsak strežnik ima nameščen **strežniški certifikat** s katerim izkazuje **javni ključ**, prav tako se hrani **zasebni ključ**.
- Dovoli brskalnikom, da vzpostavijo varne seje s spletnimi strežniki. **Brskalniki lahko imajo svoj certifikat, ni pa nujno**.
- Vrata **443** so rezervirana za HTTP prek TLS / SSL in s temi vrati se uporablja protokol **HTTPS**.
- **Druge aplikacije**: IMAP prek TLS: vrata 993, POP3 prek TLS: vrata 995
- Sestavljen je iz treh **pod-protokolov** na višji ravni:
  - **TLS Handshake**
  - **TLS Alert**
  - **TLS Change Cipher Spec**

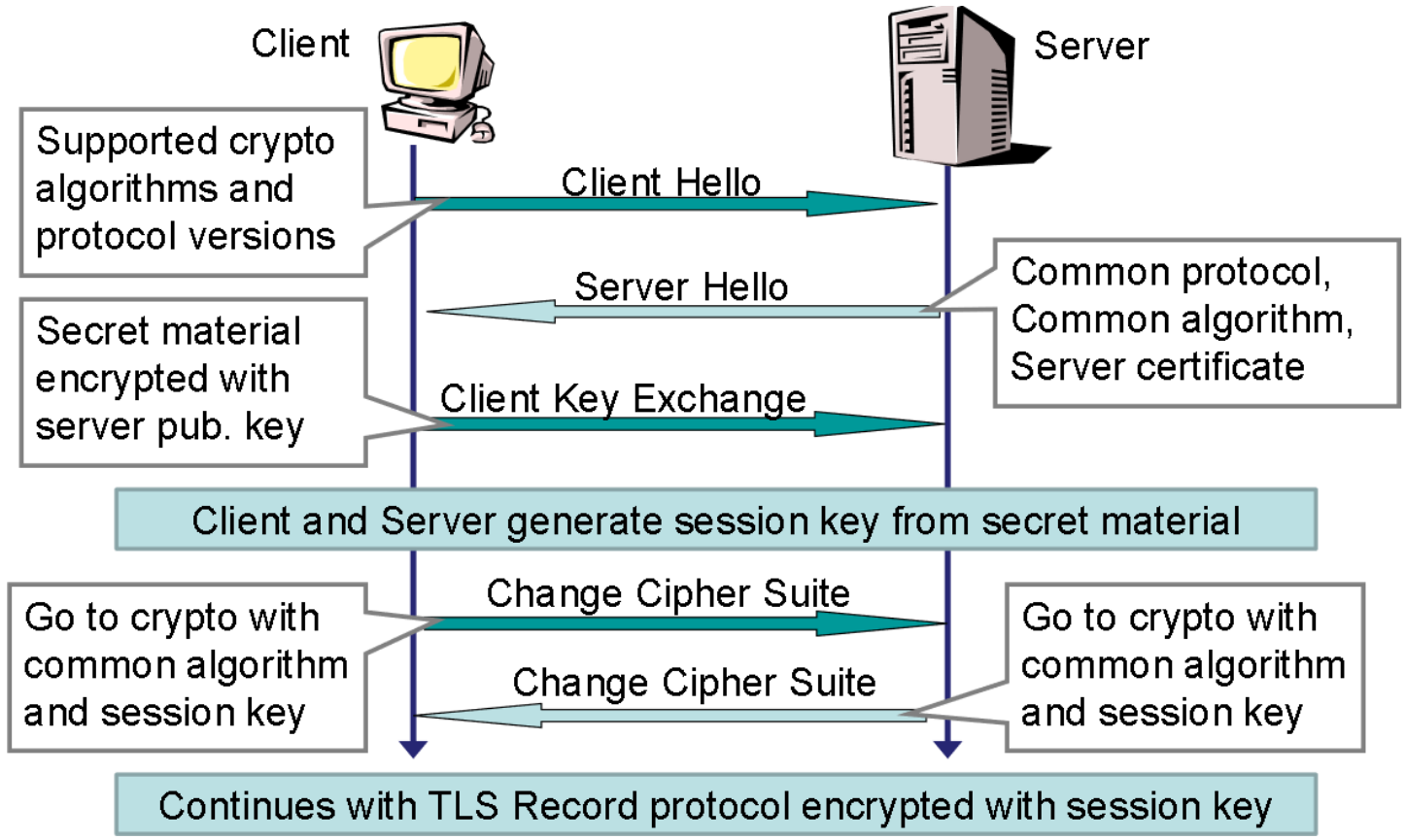


Protokolni sklad

Vir slike: Nils Gruschka, IN2120

TLS rokovanje (handshake)

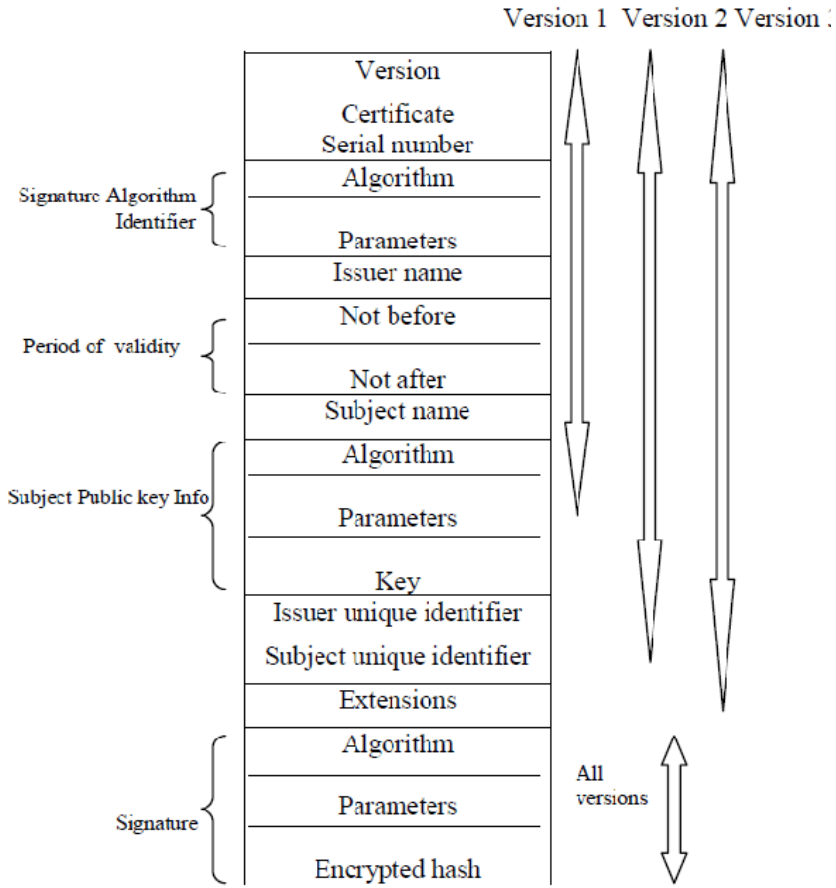
- Verificiranje **X.509 certifikata strežnika** (preko javno dostopnih baz – ti. **Avtoritete certifikatov, CA**), opcijsko tudi brskalnika (uporabnika)
- **Izmenjava o podprtih TLS protokolih**, asimetričnih (RSA, DH), simetričnih šifrirnikih (AES128, AES256 itd.) in kriptografskih zgoščevalnih funkcijah (SHA256, SHA384, …)
- **Zakaj asimetrično** (pri izmenjavi ključev) in simetrično šifriranje (pri prenosu podatkov)?
- Podrobnost protokola: https://tls.ulfheim.net/



Vir slike: Nils Gruschka, IN2120

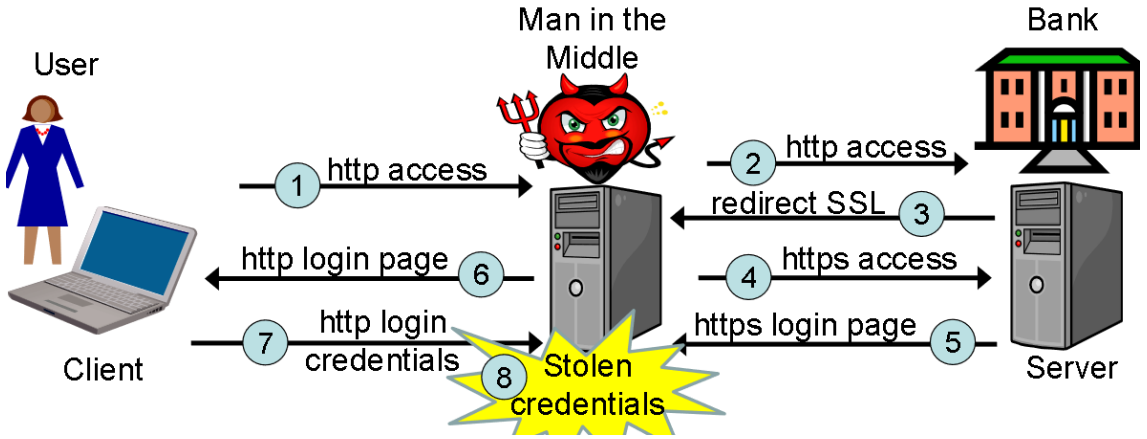
X.509 certifikat

- Najbolj razširjen standar za **hranjenje privatnih in javnih ključev**.
- V veljavi že od 1988 s strain org. ITU-T.
- Možno **verižno podpisovanje**.
  - **Preverimo** npr: https://feri.um.si
  - Poglejmo seznam CS v brskalniku



TLS izzivi

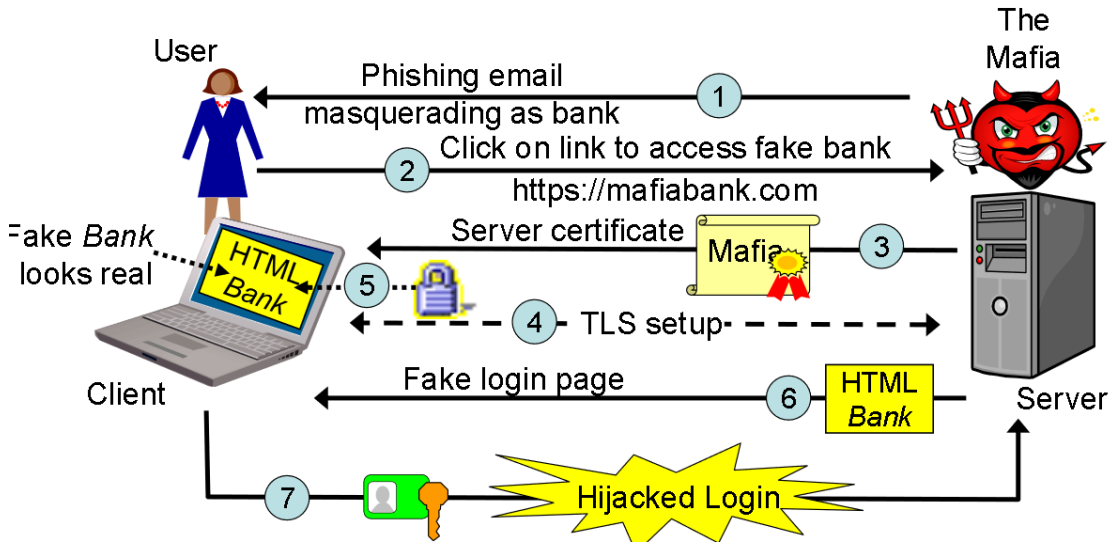
- Možna prestrežba **rokovanja z MITM napadi** (npr. preko **HTTPs ti. stripping**)
  - **Preventiva na strani brskalnika**
- Za **TLS protokolu je možna ranljivost** → redno posodabljajte odjemalsko in strežniško programsko opremo
- Tudi ranljivosti **v kriptografskih algoritmih** → konfigurirajte strežnik, da izključi šibke algoritme
- TLS zagotavlja varnost samo za **eno povezavo TCP** - brskalnik lahko vzpostavi povezave **HTTP in HTTPS**; celo **na isti strežnik**
- Uporabnika se lahko prevara, da **namesto HTTPS uporabi HTTP**
- **Ponarejeno potrdilo korenskega strežnika** lahko ogrozi vsa potrdila, npr. Lenovo Komodiaadvarescam



Vir slike: Nils Gruschka, IN2120

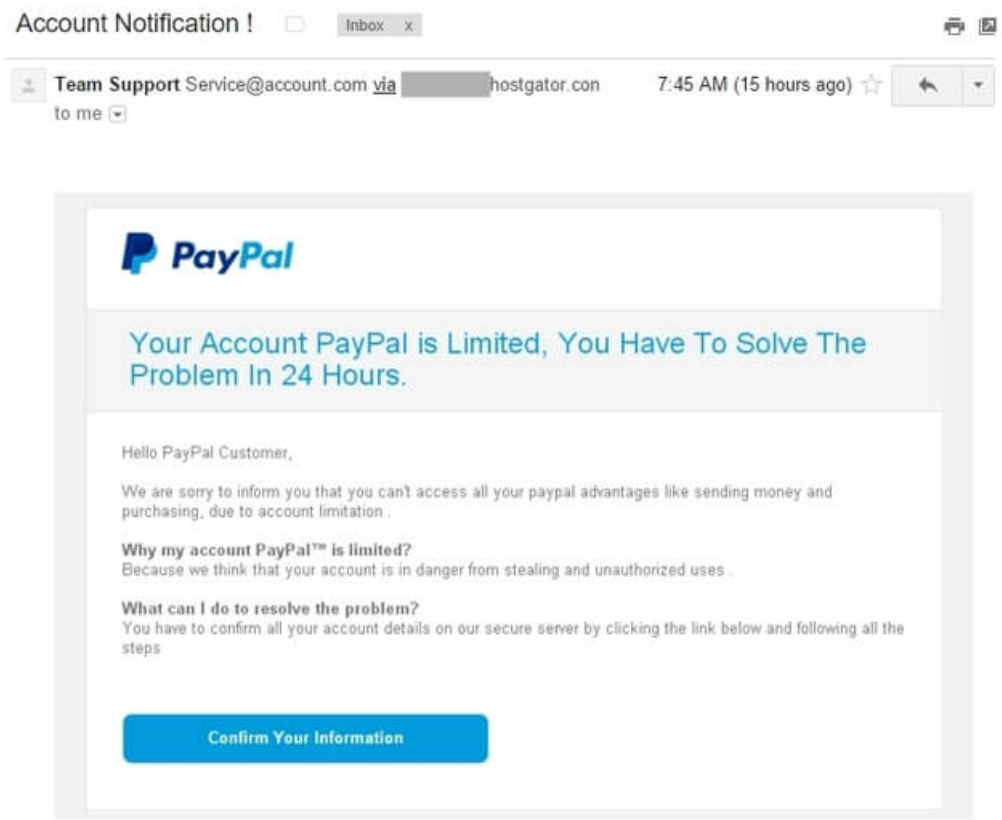
TLS MITM preko HTTPS in Phising napadi

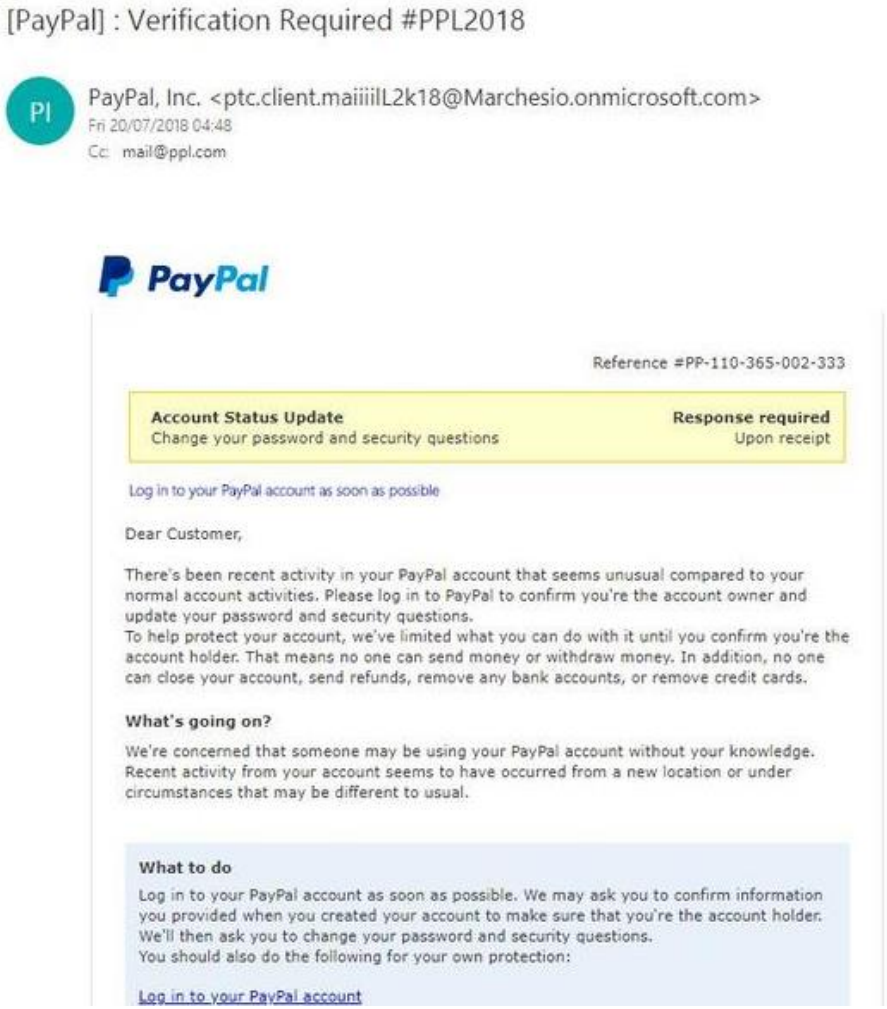
- **Veljavna strežniška potrdila ni težko pridobit**.
- **Storitve CA** za pridobivanje strežniškega certifikata:
  - **Letsencrypt**
  - AWS Certificate Manager
  - ServiceNow Now Platform
  - Microsoft BitLocker
  - VeraCrypt
  - FileVault
  - Sophos SafeGuard Encryption
  - Boxcryptor.
  - ...
- Zanimivo branje:
- Over 14,000 SSL Certificates issued to PayPal phishing sites.
  - https://www.thesslstore.com/blog/lets-encrypt-phishing/



Vir slike: Nils Gruschka, IN2120

TLS MITM preko HTTPS in Phising napadi





Primer 1. kolokvija

3. Če izvedemo uravnoteženje bremena nad HTTP prometom, imamo prednost zaradi ? **[1 T]**
(a) Horizontalnosti (b) Naprednega filtriranja (c) Vertikalnosti (d) Predpomnenja podatkov

4. Naštejte dve težavi pri vtičnikih (plugins) v spletnih brskalnikih. **[2 T]**

---

5. JavaScript pogoni uporabljajo Inline caching (IC) pimarno pri ? **[1 T]**
(a) Instanciranju objekta (b) Klicu funkcije (c) Brisanju objekta (d) Alokaciji pomnilnika

6. Na podlagi katerega protokola deluje geografsko uravnoteženje prometa? **[1 T]**
(a) UDP (b) DNS (c) TCP (d) CDN

11. Preko katerega protokola je uravnoteženje bremena statičnih strani najhitrejše ? **[1 T]**
(a) HTTPS (b) UDP (c) HTTP (d) TCP

12. Naštejte dve strategiji za uravnoteženje bremena spletnih strani. **[2 T]**

---

13. S katero metodo lahko vnaprej generiramo statične spletne strani na strežniški strani ? **[1 T]**
(a) Prefetching (b) Prerendering (c) Pregenerating (d) Preloading