- injectioni se zgodijo, ko ne ločimo vnešenih podatkov od logike:
  - `SELECT ...`
  - `PING ... {userdata}`
  - moramo sanitizirati stringe, ločiti podatke od logike, uporabiš ORM

XSS:

- pri SQL injectionu napadaš strežnik, želimo na strežniku dobiti podatke, ga podreti ...
- pri XSS napadamo cliente, ko se naš JS izvede na brskalniku od clienta
- user input `<b>burek</b>` lahko damo noter v stran kot del strani in se bo to dejansko izvedlo v brskalniku (sicer si želimo, da se bo dobesedno user input izpisal, brez da se izvede):
  - lahko damo noter script
  - `"<script>...</script>` - če damo na začetek `"`, bomo po možnosti zaprli kakšen string za user input
- če smo server ownerji nam je vseeno, če clientu ukrademo token, ker smo ga itak mi zgenerirali
- če nekdo zunanji da nek script na naš server in tako dostopa do vseh secretev:
  - cookie
  - localStorage
- script se bo izvajal na mašini od uporabnika - moramo si ga nekako poslati do sebe:
  - server zlorabimo kot API - uporabnik sploh ne bo opazil
  - ali pošlji podatke na napadalčev server

Stored XSS:

- shranimo podatke (našo skripto) na bazo in server to streže

Reflected XSS:

- uporabniku pošljemo nek url, ki ima script v url parametru
- to gre na server, ki lahko ta query parameter nariše direktno na spletno stran in izvede script, ki je bil v parametru
- stored je bolj problematičen, ker bo imel problem vsak, ki pride na to spletno stran; pri reflected rabiš vsakega posebej prepričati

WEB3:

- v app.py pride invitation iz query argumentov - reflected XSS

- `/api/new` imamo POST request, da gredo podatki v body, kjer lahko napišemo veliko več podatkov
- `localhost:8080/new?invitation=<b>123</b>` - bold text se dejansko izvede
- `localhost:8080/new?invitation=<script>alert("hello")</script>`
- `localhost:8080/new?invitation=<script>alert(document.cookie)</script>` - dobimo session ID od PHP direktno, napadalec nam lahko ukrade cookieje, session in se na strani pretvarja da je nekdo drug (v njegovem imenu pošilja requeste)
- stored XSS:
  - damo noter v new post `<b>123</b>`
  - če damo `123<script>alert("hello")</script>`, se script taga v HTML sploh ne bo vidilo, ampak se bo samo izvedel; hkrati bo post zgledal čisto normalno
- rešitev:
  - sanitiziramo input z `escape(...)` (`from html import escape`) ali `urllib.parse.quote(...)`
- uporabljaj Http-Only cookieje - browser jih bo uporabil samo, ko dela HTTP request - JS nima dostopa do njih
- lahko damo Content-Security-Policy, da povemo, kam se lahko povezujejo scripti
- CSRF token:
  - v primeru, da zahtevamo nekaj na API, dobimo one time use token, da dokažemo, da smo to res mi
  - ne moremo iz ene random strani narediti requesta na API

WEB3 part 2:

- za slike se downloadajo v /images direktorij in se iz tam potem hosta
- server požene `curl`, ne direktno mi
- veliko servicov to dela, da npr. cacha slike, konpresirajo sliko, zbrišejo metapodatke o sliki
- problem je, da uporabljamo curl, ki ima še ogromno drugih možnosti
- lahko rečemo `curl localhost:8080/images/https://google.com` - lahko uporabimo ta strežnik zato, da napadamo druge service in bo izgledalo kot da ga napada naš hijackan strežnik
- lahko rečemo `curl localhost:8080/images/file:///etc/shadow
- temu rečemo SSRF - server side request forgery
- običajno strežniki niso sami v mreži - lahko raziskujemo lokalno mrežo
- imamo samo GET request
- če je server v bistvu v cloudu, ima cloud server neke metadata serverje, ki imajo razne authentication tokene, s katerimi lahko dostopamo do nekih internih zadev, če več serverjev laufa na istih credentialih:
  - sedaj imamo GET request + credentials => lahko več stvari ukrademo

- zaščita:
  - ne kliči direktno shella in programov v njemu, ampak uporabi kakšen library, ki naredi to stvar
  - pri nas namesto `curl` lahko uporabiš `requests` library
  - lahko pogledamo, kakšen tip podatka dobimo nazaj od requesta - če ni slika, ga zavrnemo - kako bi tu onemogočili dostop do mreže:
    - omejimo, da pustimo samo requeste od internal ip-jev
    - po mreži lahko skačejo samo requesti od internal ip-jev

```
ip = resolve(path) # nevarno če tu dobimo internal ip...
if ip.internal:
    error
return requests(path) # ... in tu šele ugotovimo, da je private
```

-
  - ker so ip-ji lahko napisani na različne načine, ali pa domene, moramo najprej resolvati to
  - če gledamo kakšne domene, ki se večkrat spreminjajo, je time of use, time of check drugačen
  - domene imajo TTL za caching
  - če TTL nastavimo na 0, lahko naredimo dve skripti - ena, ki čekira IP domene in ena kot smo ja napisali zgoraj - pri `ip` bomo dobili public ip, do konca kode pa bo resolvalo, da je private in bomo obšli tisti if stavek
  - če uporabimo nek library, so take stvari pogosto že popravljene