- TOCTOU:
  - najprej preverimo, če imamo dostop do nečesa in šele potem dostopamo do tega vira - vmes lahko izgubimo pravice oz. vir ni več dostopen (če se vmes zgodi context switch/se začne drug thread izvajati)
- hkratni dostop (race condition):
  - lahko dobimo čudne/nekonsistentne rezultate, ko dva procesa dostopata do istega vira
  - npr. dva procesa uporabljati isti števec in ne veš, kdo ga bo prvi povečal
- side channel:
  - pridemo do informacij do katerih ne bi smeli, ampak uhajajo iz nekje drugje (npr. porabe elektrike, porabe časa)

Defenzivno programiranje

Napad:

- najdemo ranljivost (ponesreči ali namenoma)
- izkoristimo ranljivost:
  - dobimo podatke
  - dvignemo privilegije

Obramba:

- uporaba specifičnih jezikov (Rust, Go):
  - del odgovornosti porinemo na nivo izbiro jezika - nekaterih napak ne bomo gledali, ker v tem jeziku niso mogoče
  - preprečiti ti določene vzorce, ki so nepravilni (ki niso mišljeni za ta jezik)
- rokovanje z izjemami
- sanitizacija argumentov
- IDE nam lahko zazna napačne vzorce
- omejevanje privilegijev:
  - najlažje je reči, da imajo vsi dostop do vsega in potem tako ostane
- happy path - naredimo samo tako, da dela
- konsistenca in disciplina:
  - držiš se dobrih praks in ne delaš začasnih rešitev, ki postanejo trajne
  - vse je odvisno od tvojega standarda koliko pravil se boš držal

Programsko okolje:

- koda je berljiva, sledi nekim vzorcem
- preveriš stvari za sabo

Dobra praksa in vzorci:

- kodo, ki je bila že spisana, želimo generalizirati, da imamo neko prečiščeno kodo, ki dela
- preverba vrednosti - ali je input res tisto, kar smo mislili, da je
- avtomat (state machine):
  - vedno vemo, kako smo prišli do nečesa
  - katera stanja obstajajo?
  - če pridemo do neveljavnega stanja, moramo vedeti, kako smo prišli do njega in izpišemo koristen error
- pisanje dokumentacije

Zakaj je vzdrževanje takšnega načina težko:

- težko se je držati pravil
- včasih time of delivery trpi zato, da imamo lepšo kodo (kasnejši production)
- legacy koda je lahko slaba - je ni vedno dobro ponovno uporabiti
- vedno rabimo predvideti vse možne inpute
- ko se zgodi napaka, uporabniku ne smemo izpisati error message-a v debug načinu

Dobre prakse:

- dober QA team
- temeljiti testi
- profiling - spremljanje časa delovanja
- strogi standardi kodiranja
- ne prezgodaj optimizirati kode - najprej hočeš, da stvari delajo, potem optimiziraš
- predvidljiva programska oprema:
  - ne pridemo v čudna stanja, ki ni definirano, kaj se mora zgoditi
- varno rokovanje s števili:
  - da so števila res števila in da so taka, kot pričakujemo
- uporaba tipiziranih spremenljivk
- sanitizacija vhoda

Varno rokovanje s števili:

- uporaba pravilnih tipov
- npr. v C ne tlačit `long` v `int`

Varnost pomnilnika:

- preverjanje index out of bounds
- štetje referenc (referenca je pointer na pointer), da lahko kazalec sprostimo, ko ni več nobenih referenc

Primer file upload:

- `f.resolve()` ti da absolutno pot, da ne moreš napisati `/app/data/../../etc/passwd`, ker se ti bo to resolvalo v `/etc/passwd`, ki se ne začne z `/app/data` - rešili smo path traversal
- preverimo samo `/app/data`, torej imamo dostop tudi do `/app/database.sqlite3` - rešitev: preverimo, da se pot začne z `/app/data/`
- nočemo napak iskati v runtime, ampak čim več v compile time - compiler z vsemi warningi
- nevarna hramba tipov:
  - v binarno datoteko hranimo podatke o nekem classu - ohranimo vse tipe, metapodatke ipd.; noter damo lahko bolj kompleksne stvari kot v JSON
  - v binarni datoteki lahko shranimo npr. funkcijo; potem ko jo naloadamo jo lahko kličemo
  - če se ne zavedamo točno, kaj je v binarni datoteki shranjeno, so lahko problemi
  - lahko si napadalec v binarno datoteko da neko svojo funkcijo, ki se bo potem izvedla

Primer Palo Alto:

- če si v firewall poslal request z npr. /bin/sh, se je to loggalo, da se je zgodil potencialen napad
- ko se je delala analiza teh loggov, je ta koda escapala in si lahko dobil access do firewalla (ta cronjob za analizo loggov je tudi tekel kot root)

Log4J:

- user input lahko uporabiš kot format, podobno kot `printf(user_input);`
- v JNDI lahko povemo, da naj gre datoteko iskati nekam na splet in to se bo poslalo kot format, torej lahko preneseš neko zlonamerno datoteko, ki se bo pognala na strežniku