- defenzivno programiranje - preveriš za vse možne napake npr. checkanje izven ranga, preveriš, če se kakšna funkcija nepravilno zaključi; preverjamo npr. če pri množenju ne pride do overflowa, catchaš exceptione (preveriš, če je prišlo do exceptiona in ustrezno odreagiraš nanj)

Varnost sistemov:

- kaj narediti, če nisi vsega sprogramiral sam
- omejitev dostopa:
  - ne želimo, da ima neka aplikacija pravice za dostop do vsega
  - lahko nekemu procesu dodelimo, kaj je njegov root, npr. /var/www/
- ko uporabimo nek software v našem produktu, pogledamo na kakšne vzorce je sprogramirano, kateri jeziki so uporabljeni
- ko imamo več programerjev, se zmenimo za vzorce, uporabljene jezike, kako bomo delali exception handling
- gremo unit testat naš produkt
- če smo mi programerji se zmenimo katera orodja bomo uporabili (IDE, plugini)
- omejevanje privilegijev:
  - preventivno omejimo procese, netwroking na procese, ki laufajo na serverju
  - ne želimo, da se mešajo stvari drugega SW in SW, ki smo ga mi napisali
  - omejimo človeka, ker je lahko žleht ali nesposoben, zanemari neke aspekte varnosti

Omejevanje:

- aplikacije damo v peskovnik, da ne morejo spreminjati sistema
- že tako ali tako veliko stvari leti ven prek oglasov, prodajanje podatkom oglaševalskim agencijam
- danes je veliko aplikacije web
- z vsako npr. posoditvijo OS dobimo nove možnosti omejevanja pravic aplikacijam
- peskovnik za proces:
  - premaknemo root procesa
  - omejimo sistemske klice, ki jih proces lahko izvaja (funkcije, ki jih OS rabi za delovanje)
- naredimo control groupe in namespace in omejimo na nivoju virtualizacije - kontejnerji
- tudi docker kontejnerji imajo ranljivost in CVE-je, da nam nekdo ven skoče iz kontejnerja ali iz virtualke
- neskončen boj s skokom iz virtualizacije

Peskovnik znotraj procesa:

- aplikacija omejuje kodo uporabnika
  - V8 Javascript engine za hitro renderanje strani, vsi browserji ratajo bolj v skladu s standardi za pisanje in renderiranje strani - koda se prevede v vmesno kodo in potem interpretira znotraj tega V8 peskovnika
- če nam uspe skočiti iz peskovnika, dobimo dostop do matičnega sistema:
  - vedno, ko poskušamo delati optimizacijo hitrosti, delamo tradeoff z varnostjo

Peskovnik za proces:

- omejimo sistemske klice za proces (write, read ...)
- lahko omejimo dostop do networka (poleg firewalla za vse procese imamo lahko še specifična pravila za posamezne procese, da omejimo dostop do networka na nivoju procesa)
- omejimo dostop do podatkov
- za tracking sistemskih klicev: `strace`
- sistemski klici:
  - `exec` - nadomesti kodo enega procesa z drugo kodo (ne naredi novega procesa - ostanejo file descriptorji, memory allocation)
  - `fork` - naredi nov proces

Seccomp:

- omejuje sistemske klice, lahko napišemo pravila, kaj eksplicitno dovolimo/ne dovolimo
- če proces skuša uporabiti nek klic, za katerega mu nismo dali pravic, se bo ubil
- ideja:
  - program omeji samega sebe
  - preden naredimo seccomp_init lahko delamo karkoli želimo, od tam naprej pa se program drži pravil iz seccomp_rule_add
  - če nekdo najde npr. buffer overflow exploit v browserju, ne bo imel pravic za exec /bin/bash
  - v init povemo, kaj naj naredi s procesom, ko kliče nek sistemski klic - lahko ga ubijemo ali pa kakšno drugo akcijo naredimo
  - lahko omejimo tudi argumente sistemskih klicev s SCMP_Ax (x je številka argumenta)
  - lahko npr. za write omejimo samo na std out do 64 znakov
  - tudi če nekdo dobi remote code execution, mora slediti tem pravilom in ne more delati ravno česarkoli hoče
  - programi, ki se execajo/forkajo iz tega procesa, imajo prav tako ta seccomp pravila in so enako omejeni glede sistemskih klicev - moramo dovoliti fork in exec (mogoče

samo exec) klica
- pomankljaj: omejeni smo z zelo low level pravili (ne moremo npr. primerjati stringov)

Landlock:

- malo bolj konkreten resource manager
- omejitve s stališča file systema in networka
- lahko delamo bolj advanced filtre
- ko čekiramo file in imamo TOC/TOU ranljivost, lahko med TOC in TOU datoteko spremenimo v symlink in bomo pisali nekam drugam
- z Landlock moramo imeti dostop do točno tega file descriptorja, zato se ne mora spreminjati vmes
- vse to laufa znotraj kernela - vsi procesi, ki se execajo ali forkajo, bodo imela vse ta Landlock pravila - novi procesi lahko samo dodajajo bolj ostra pravila, ne morajo jih relaksirati
- fork staršu vrne PID otroka, otrok pa dobi PID 0
- PID starša se zmanjša za 1
- če se želimo znebiti fork bomba, moramo omejiti število forkov, ki jih en proces lahko naredi
- dedujemo file descriptorje, permissione; file descriptorji ostanejo odprti
- file descriptorji 0, 1, 2 - stdout, stdin, stderr
- med forkom v otroka skopiramo samo tisto, kar bo spreminjal, ker nas kopiranje stane in ga želimo minimizirati (optimizacija na nivoju copy on write)
- niti si delijo naslovni prostor (heap in stack; stack ima v bistvu vsaka nit svoj), imajo manj overheada za kopiranje od procesov - hitreje jih spawnamo
- proces pooling, da je ceneje delati s procesi - namesto, da procese ves čas spawnamo in ubijamo, med njimi preklapljamo
- imamo worker threade, ki so del istega poola

Virtualizacija:

- control groupi:
  - omejitev dostopov do CPU, pomnilnika
  - omejimo, do česa ima zadeva dostop in bo izgledalo, kot da smo na drugem sistemu, čeprav ampak v ozadju laufa isti kernel
  - s control groupami omejimo koliko katerega vira lahko kdo ponuca
  - vsak proces bo videl cel memory - s control groupami npr. preprečimo, da bi en proces poskusil porabiti cel RAM iz sistema, ampak lahko samo nek omejen del porabi (zdi se mu, da ima sistem samo toliko RAM-a vse skupaj)
  - CPU quote in memory quota - koliko CPU in koliko RAM-a lahko proces porabi

    - cifre v quota niso nujno cele - lahko rečemo, da proces porabi npr. 50% CPU - lahko bolj podrobno omejimo dostop do CPU kot z virtualko (ko dela proces scheduler in vidi, da je en proces že porabil ves svoj čas, ga bo schedulal za kasneje)
    - v virtualki lahko nastavimo samo koliko bytov RAM-a porabi, koliko threadov porabi
    - ne rabimo vedno cele virtualne mašine s celim OS, ki bo rabila GB RAM-a, ampak lahko naredimo manjšo enkapsulacijo s kontejnerji, ki vključuje samo to, kar želimo laufati - porabijo MB RAM-a
- namespaci:
    - virtualne naprave

Kontejnerji:

- na nek način laufa celo virtualko, ampak bolj omejeno
- dobro je, da ne rabiš delati celega setupa, ampak kontejner samo zbuildaš in poženeš (običajno) - lažje je postaviti peskovnik kot pa da gremo na roko vse omejiti; to nas seveda malo stane glede performanca
- kot male virtualke, ki porabijo manj resursov kot če bi cel OS postavili, ker ne rabimo zares celega OS, ampak želimo izvajati samo neko specifično aplikacijo

Docker

- lahko laufamo različne tipe OS v dockerju
- v dockerju imamo čisto drug sistem, kot na matični napravi; cel OS je postavljen ločeno
- ima svoj file in network namespace - vidi svoj datotečni sistem in svoj network
- docker container dobi svoj namespace, svoj IP in se obnaša kot virtualka, ločena od sistema
- zadevo na host sistemu vidimo v seznamu procesov, vidimo programe, ki jih laufa
- lahko zalaufamo container z nekimi pravili seccompa, z nekimi control groupi
- syscalle bo kontejner delal na host kernel, zato ga vidimo samo kot en dodaten proces
- lahko uporabljamo druge vire od host sistema:
    - naredimo, da ena mapa kaže na mapo od host sistema - file system je v ozadju isti, samo kontejner ga ne vidi, če ne skonfiguriramo drugače
- veliko manjši memory in CPU footprint
- uporablja iste resource kot host machine
- če nekdo napade in zlorabi en kontejner, bo še vedno zaprt znotraj kontejnerja
- uporablja isti kernel (sistemske klice dela na isti kernel), zato ima manjši memory footprint; pri virtualkah rabimo še dodatno virtualizirati kernel in se potem sistemski klici delajo na ta virtualiziran kernel, ne na kernel matične naprave
- če ima kernel exploit, lahko pridemo do host kernel (npr. Dirty COW: dirty bit od copy-on-write exploita, da lahko pišemo v druge datoteke - pod pravim race conditionom smo lahko

iz kontejnerja pisali po host machini)

- doda memory overhead, ampak je dober tradeoff z omejitvijo aplikacij v peskovnik
- ko si na Linux, imaš lahko samo Linux kontejnerje, če si na Windows, imaš lahko samo Windows kontejnerje, če nimaš WSL, ker kontejnerji uporabijo isti kernel in morajo biti z njim kompatibilni