Peskovniki:

- omejimo kaj proces lahko dela
- na web imamo V8
- če hočemo več procesov enkapsulirati, naredimo kontejenerje, virtualno napravo (virtualizacija)
- sistemski klici:
  - je način komunikacije z jedrom OS
  - ko želimo nekaj narediti, kar zahteva jedro OS, bo OS določil ali to dovoli ali ne
  - ko želiš odpreti datoteko, narediti nov proces, končati proces, moraš narediti sistemski klic
  - je način komunikacije z jedrom OS, prek katerega OS omogoča dostop do posameznih funkcionalnosti OS in nadzira, kateri procesi imajo dostop do posameznih funkcionalnosti (kot so branje datotek, ustvarjanje procesov ...)
- namen virtualizacije:
  - enkapsulacija
  - lažji nadzor privilegijev (več procesov damo iste privilegije)
  - da pokupčkamo stvari v nek predal, kjer jih lažje nadziramo

Raz-zbiranje (dissasembly):

- če nimamo dostopa do izvorne kode, če aplikacija ni naša
- imamo izvršilno datoteko - jo bomo razpakirali:
  - običajno izgubimo berljivost
  - manjka en kup metapodatkov, ki jih rabimo preden scompilamo kodo
- ko nekaj ne dela oz. če ima nek malware lahko probamo ugotoviti, zakaj tako dela/ne dela
- ko nekaj decompilaš imaš še dolg proces, da pravilno poimenuješ spremenljivke, popraviš tipe; konstanten proces da lepšamo kodo, da postane berljiva

Kako se borimo:

- antivirusi:
  - če imamo en del izvršilne datoteke (podpis), lahko pogledamo, če je kdo ta podpis že identificiral
  - ali sem to kodo že kje videl
- firewall
- pasivno:
  - pride nekdo, ki ima nek virus na računalniku

    - moramo delati analizo kako je do tega prišlo, človeški vidik varnosti
    - razen v enterprise ni nek usmerjen napad, ampak so script kiddie ali pa nek scam mail
    - da gledamo kaj se dogaja, lahko damo v nek peskovnik
    - delamo statično in dinamično analizo
    - tega ne delamo na primarni napravi
    - kako se izvaja, kako se širi, ali obstaja killswitch
    - običajno od zlonamerne programske opreme hočemo nekaj dobiti

Primer:

- Cisco Runner in Line Dancer:
    - malware, ki teče samo v RAM
    - ko zazna, da boš rebootal sistem, se začasno skopira na disk
- malware ima ponavadi veliko zakrivanja, kaj dela in da se zablokira, če ga proba nekdo debuggat, da je težko opazovati kako se izvaja
- primer:
    - Volkswagen je naredil, da so rezultati testov passali, ko so šli čez unit teste, ko si jih zares testiral pa je failalo
    - zadeva ugotovi, če jo probamo testirati (če teče v testnem okolju) in ponaredi rezultate, da passajo teste

Fuzzing:

- kako najdemo kaj program dela
- včasih izgubimo dostop do izvorne kode in želimo ugotoviti, kako dela
- statična analiza:
    - ko smo pisali Windows 3, ni bilo dobrih IDE-jev - s statično analizo kode smo že veliko buggov najdli
    - ko je program velik, bo to počasno - kako stvari do neke mere avtomatizirati:
- fuzzing = tehnika avtomatiziranega testiranje:
    - tisto kar smo probali z unit testi pokriti do neke mere, poskusimo posplošiti
    - pri unit testih smo gledali, da pokrijemo nek primer
    - pri fuzzing iščemo, če se bo nekaj usulo pri nekem inputu - podobno kot unit testi, ampak probamo crashati zadevo
- orodje AFL:
    - dobimo sistematično generiranje semi random inpute
    - sprobamo te teste, da vidimo, če nam kaj crasha - dobro za low level stvari, kot so kerneli, compilerji, image processing v browserjih
    - če fuzzer teče npr. pol ure in ne najde crasha, potem lahko rečemo, da je vse OK

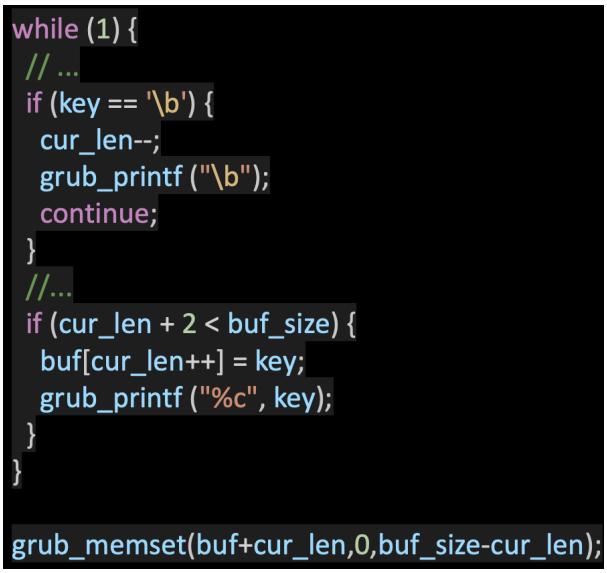
    - včasih se mora veliko stvari mora poklopiti, da pride do crasha, ampak še vedno bo fuzzer to najbrž hitreje našel kot mi - najdemo kakšne morebitne memory leake

Analiza binarnih programov:

- želimo stestirati stvari kot sysadmin, kot zunanji izvajalec
    - Namesto naključnih vhodov analiziramo program in zgeneriramo vhod
    - Rekonstrukcija grafa izvajanja
    - Automatizirano tudi iskanje iz izkoriščanje varnostnih lukenj
- angr:
    - mu rečemo, da želimo priti do nekega assembly ukaza in nam sledi kako smo prišli do njega
    - sledimo stanju programa (memory state)
    - "najdi način, kako boš prišel do te funkcije"
    - lahko pridemo mimo checkou, primerjanj; na malware najdemo killswitche
    - dolgo časa teče, da najde kako priti mimo nekega checka - to dela stabilno z neko logiko, da predicta branche
    - določene osnovne primere zna exploitate, da pride do win funkcije
- to ni isto kot fuzzing - fuzzing vzame neke vhode, za katere vemo, da delajo in doda malo random simbolov, permutacij, da dobimo bolj random inpute in vidimo, kako se program odzove (ali crasha - crash običajon pomeni, da lahko nekaj exploitamo)

Primer GRUB:

- backspace je znak:
    - ko zmanjšamo curr_len, ne pogledamo, če je manjši od 0 - lahko gremo v neskončnost nazaj in v nadaljevanju lahko pišeš po underflowavem delu spomina

- 

```
while (1) {
  // ...
  if (key == '\b') {
    cur_len--;
    grub_printf ("\b");
    continue;
  }
  //...
  if (cur_len + 2 < buf_size) {
    buf[cur_len++] = key;
    grub_printf ("%c", key);
  }
}

grub_memset(buf+cur_len,0,buf_size-cur_len);
```

- če smo naredili 28 backspaceov, smo lahko prešli password check, ker smo prišli v grub recovery mode od koder lahko ročno zaženeš/mountaš OS
- fuzzerji so uporabni, ker nam avtomatsko grindajo možnosti inputov, da nam ni treba tega na roke delati