Return oriented programming

- izvajamo veliko majhnih kosov kode (gadgets), ki spremenijo neke registre in jih sestavimo skupaj v sebi dopadljivo zaporedje
- manjša kot je kocka, več stvari lahko zgradimo z njo, lahko jo reusamo

Gadgets:

- `leave` in `ret` ukaza
- leave:

```
mov rsp, rbp
pop rbp
pop rip
```

- base (frame) pointer nam pove naslov prejšnjega stacka, da lahko popravimo stack pointer nazaj
- 1. korak: rsp premaknemo na mesto, kjer je kazal rbp
- 2. korak: dobimo nov rbp s pop
- 3. korak: ret addr se zapiše v rip, da potem skočimo na ta naslov (PC se nastavi na rip, ne PC=PC+1)
- imamo mnogo "pop/ret" kombinacij
- v posamezne registre lahko vstavimo poljubne vrednosti:
  - če ima neka oseba dostop do nečesa, je to najbrž neka vrednost v registru
  - nastavimo poljubno vrednost in namesto da skačemo na win funkcijo, si jo lahko sami sestavimo iz gadgetov
  - npr. funkcije, ki nastavi moj account na admin ni, ampak jo lahko sestavimo iz gadgetov
  - ne skoči na win funkcijo ampak skoči na kos kode, naredi kar želiš in skoči na naslednji kos kode in ponavljaj
- če bomo veliko gadgetov sestavljali skupaj, bomo verjetno kaj polomili
- zelo zamudno sestavljanje

Primer:

- nikjer se ne kliče funkcija win
- prejšnji teden:

    - prepišemo canary in v ret addr napišemo naslov win funkcjie, ki ga dobimo za disassembly
- vemo, kateri argument funkcije je na katerem registru (če bi bilo preveč argumentov, bi šli na stack)
- https://syscall.sh
- 

- syscall-i so v tabeli, torej če bi enega hoteli pobrisati, bi morali vse offsete spremeniti (vse vrstice zamakniti gor; x64 ima manj syscallov, ker je bilo itak vse treba spremeniti in smo ven pometali še stvari, ki jih ne raibmo

Izogibanje ROP:

- ASLR:
    - malo pokvarimo pomnilniško lokalnost; naslovi funkcij niso vedno isti in potem ne vemo, kam točno moramo skočiti
- preverjanje, kam skačemo (kot canary)
- SEHOP:
    - ne dovolim ti, da greš in mi prepišeš tisto vrednost
- blind ROP:
    - poskušamo crashati proces in ko se to zgodi, lahko dobimo neko vrednost v trenutku (dumpfile, vrednosti registrov, stanje procesorja)
    - zelo zamudna stvar, ker so programi ponavadi kompleksni in mogoče se ga ne da crashati
- običajno to delamo s skripto, ne na roke

Spletna varnost

- veliko tehnologij, ki jih ne razumemo v celoti
  SQL napadi:
- SQL injection:
    - v SQL stavek damo noter nek svoj stavek
    - najbolj nas zanima, da dobimo dostop do zalednega sistema
- včasih želimo samo kaj več zvedeti o sistemu:
    - če ugotovimo kakšen sistem je zadaj, lahko probamo znane exploite
- nameni:
    - kraja identitete
    - uničenje podatkov
    - nedostopnost storitve

SQL injection:

- problem sanitacija nizov
- uporabi ORM, da ti pravilno pripravi statemente, ali vsaj escape string

Zakaj pride do tega:

- malomarnost, nimamo časa, treba na hitro narediti
- neznanje
- ne vzdržujemo v redu,
- je nek začasni projekt, ki ni bil nikoli zares dokončan
- če hočemo vse podatke, damo 1=1 in npr. dobimo seznam produktov, ki še niso na voljo, ampak bodo
- union-based SQL injection (OR stavki)
- error based SQL injection:
  - dobimo še zraven dodatne podatke o bazi, da je developerju lažje odpravljati napake
  - dobimo stack trace
  - dobimo kodo okoli tistega, kjer se je pokvarilo
- včasih me zanima koliko časa se je neki izvajalo in ali se je sploh izvedlo, ne nujno kaj smo dobili nazaj

Izvedba:

- uporabniški vnos
- piškotki
- preko HTTP headerja

Union-based:

- rabimo vedeti vnaprej imena stolpcev, število stolpcev, imena tabel

Blind:

- second-order SQL injection
- ne vemo, kaj dobimo nazaj oz. nas ne zanima
- v HTTP request damo neke flage, ki so nam všeč (npr. user je admin)

Time-based blind injection:

- npr. SLEEP(15) - 15 sekund spi
- lahko preverimo verzijo sistema in če smo jo zadeli, bo sistem spal 15 sekund preden bo vrnil rezultat:

    - mogoče poznamo exploit za neko specifično verzijo sistema in se potem lahko osredotočimo na naslednji vektor napada

Pisanje datotek

- dirlisting:
    - lahko vidimo vse datoteke v nekem direktoriju, lahko gledamo neke direktorije, ki jih mogoče ne bi smeli
    - v PHP gre file upload v /var/tmp - problem, če damo nek executable in se potem to izvede na serverju
- konfiguracijske datoteke:
    - imamo neke API ključe, dostopne podatke
- v kontekstu maila najbolj pogosto - tu imaš neko datoteko in si jo naloži in izvedi

Zaščita:

- rečemo, da je root od procesa /var/www in bomo ven iz tega težje skočili (ne bomo morali npr. do /etc/passwd)
- blacklisting tipov datotek: problem, če kakšen tip pozabimo blacklistati in ga bo server še vedno zagnal
- preverimo default konfiguracije
- pogosto imamo neke dodane module, ki niso nujno varni - jih moramo izklopiti, če jih ne uporabljamo
- naložimo datoteko, jo izvedemo s klicem in rezultat dobimo na uporabniški strani

Rešitve:

- izbira naključnega imena
- vrivanje končnic (npr. vsem slik damo .jpg, tudi če je nekaj drugega)
- spreminjanje konfiguracije
- encoding imen - namesto pik damo %2E in podobno