Najbolje sigurnosne prakse prilikom postavljanja i održavanja Linux servera

Ptiček, Goran

Master's thesis / Diplomski rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:983576

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Goran Ptiček

NAJBOLJE SIGURNOSNE PRAKSE PRILIKOM POSTAVLJANJA I ODRŽAVANJA LINUX SERVERA

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Goran Ptiček

Matični broj: 45917/17-R

Studij: Programsko inžinjerstvo

NAJBOLJE SIGURNOSNE PRAKSE PRILIKOM POSTAVLJANJA I ODRŽAVANJA LINUX SERVERA

DIPLOMSKI RAD

Mentor :

Doc. dr. sc. Igor Tomičić

Varaždin, kolovoz 2023.

Goran Ptiček

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Sfera Linux servera veoma je fragmentirana zbog velikog broja mogućih distribucija i pripadnih poslužiteljskih konfiguracija. Iako je opcija mnogo, postoje generalne najbolje prakse koje se odnose na sve. Ovaj rad bavi se istraživanjem i dokumentiranjem tih najboljih praksa s naglaskom na sigurnost tako konfiguriranog sustava.

Ključne riječi: linux; sigurnost; poslužitelj; administracija; docker; firewall;

Sadržaj

1. Uvod

Iza svake web stranice ili aplikacije stoji neki server koji ju čini dostupnom. Zbog otvorene i besplatne prirode Linux sustava, on je upravo najčešći izbor operacijskog sustava svih servera. Linux distribucija možda ima mnogo, no konfiguracijskih opcija još više.

Prije postavljanja servera, treba detaljno razmotriti njegovu svrhu kako bi mogli odabrati adekvatnu distribuciju i poslužitelj koji ju podržava. Ako korisnik ima iskustva s postavljanjem i održavanjem sklopovlja, postavljanje na vlastitom računalu nudi različite benefite i veću razinu sigurnosti, no zahtjeva više posla s održavanjem.

Nakon postavljanja servera i željene aplikacije, potrebno je učvrstiti sustav. Učvršćivanje se bazira na isključivanju svih nepotrebnih funkcionalnosti i ograničavanju onih koje su potrebne koliko je to moguće. U ovom radu demonstrirano je nekoliko aspekata sigurnosti, no treba shvatiti kako sigurnost nema univerzalni recept koji, jednom implementiran, se ne treba izmjenjivati, već je nešto što kontinuirano evoluira te se treba održavati redovnim sustavskim nadogradnjama, kontejnerizacijom programa, zamjenom postojećih rješenja s modernijim i sigurnijim alternativama itd.

2. Metode i tehnike rada

U radu se uzima pristup inkrementalne demonstracije svakog koraka izvođenja neke radnje u blokovima koda. Svaki red u bloku koda koji započinje s # znakom odnosi se na izvršavanje naredbe u naredbenom retku kao *root* korisnik. Dodatno, svaki redak koji započinje s ... indicira na skraćivanje ispisa izvršene naredbe. U radu se na Linux sustav referira s *Linux*, a ne *GNU/Linux* budući da danas postoje alternativne implementacije korisničkih (*eng. userland*) programa (npr. Alpine Linux koristi Busybox). Rad većinski koristi online resurse.

3. Odabir poslužitelja i distribucije

Postavljanje Linux servera započinje odabirom samog poslužitelja. *Poslužitelj* se odnosi na samo računalo na kojem će sustav biti postavljen. Sam korisnik može biti u ulozi poslužitelja (u kojem slučaju on koristi vlastito sklopovlje (*eng. hardware*)), no češće se koristi udaljeni poslužitelj kojeg neko poduzeće iznajmljuje. Odabir distribucije isto je dio ove sekcije budući da su ti izbori međusobno povezani. Npr. neki poslužitelji nude samo nekoliko predefiniranih distribucija zbog čega ih nije moguće koristiti ako nam je potrebna distribucija koju ne nude.

3.1. Iznajmljeno sklopovlje

Korištenje iznajmljenog sklopovlja često se naziva i *Cloud hosting*. Pozitivna strana Cloud hostinga primarno je jednostavnije održavanje budući da je sav posao u smislu postavljanja sklopovlja i mreže odrađen od strane Cloud poslužitelja. Integritet podataka uglavnom se osigurava RAID10 konfiguracijom koja je apstrahirana od korisnika (korisnik vidi samo 1 disk unutar sustava).

Postoje više različitih tipova cloud hostinga. Ramnode [1] nudi:

Cloud KVM VPS
: KVM (*eng. Kernel-based virtual machine*) otvoreno je i besplatno rješenje za virtualizaciju Linux sustava. U ovom modelu hostinga, korisnik ne dobiva ekskluzivnu pristup sklopovlju, već isti dijeli s drugim korisnicima, a razina perfomansa ovisi o odabranoj tarifi. Ovo je najčešći oblik Cloud hostinga.

Virtual Dedicated Servers
: U ovoj konfiguraciji korisnik dobiva ekskluzivan pristup jezgrama procesora.

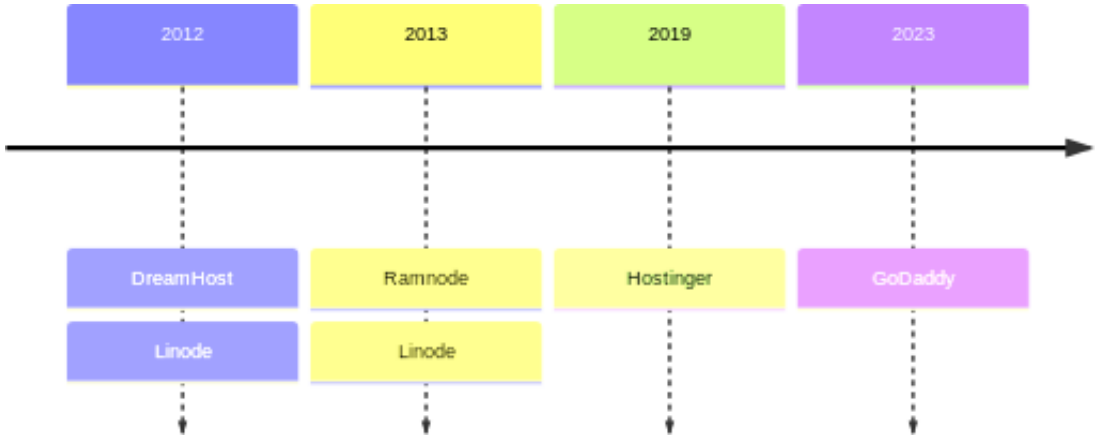
OpenVZ VPS
: OpenVZ ne virtualizira cijeli sustav kao što to radi KVM, već koristi dijeljeni sustav u kojem korisnik ima pristup samo svojem kontejneru. Svaki od tih kontejnera ima svoj vlastiti sustav, no on ne pokreće vlastitu jezgru, već koristi istu koju koriste drugi korisnici.

Odabir tipa cloud hostinga ovisi o potrebama korisnika, no svaki od navedenih tipova dijele slične pozitivne i negativne strane.

U smislu sigurnosti, pozitivna strana odabira cloud hostinga leži u činjenici da su ti poslužitelji profesionalno postavljeni u smislu integriteta podataka (RAID10), propusnosti internetske veze (optička veza), valjano postavljenog hipervizora (*eng. hypervisor*) za virtualne mašine, imaju minimalne zastoje (*eng. downtime*), no imaju i brojnih potencijalnih problema.

3.1.1. Hakeri

Budući da su ti poslužitelji popularni, često su i meta hakera koji pokušavaju dobiti root pristup virtualnim mašinama. Prije spomenuti Ramnode bio je hakiran [2] 2013. godine. Napadači su dobili pristup korisničkim informacijama i lozinkama za root pristup njihovim VPS instancama. Prema Ramnode [3], napadači su iskoristili sigurnosni propust u SolusVM virtualizacijskoj platformi koju su koristili. Ramnode nije bio jedina žrtva ovog napada, već i svaki drugi poslužitelj koji je koristio SolusVM rješenje. Ovaj napad potencijalno je imao utjecaj i na druge servise koje su Ramnode korisnici koristili budući da su lozinke bile izravno (*eng. plaintext*) pohranjene na diskovima, a isto je zbog SolusVM platforme koja nije vršila valjano sažimanje lozinke (CWE-256 [4]).



Slika 1: Prikaz nekoliko hakiranih VPS poslužitelja [2] [5] [6] [7] [8] [9]

Ova vrsta napada nije specifična cloud VPS poslužiteljima, već je potencijalni vektor napada za svaki javno dostupni server, no vjerojatnost eksploatacije je veća kod cloud poslužitelja budući da su javni i popularni te isplativija meta jer je na jednom mjestu velik broj korisnika.

3.1.2. Sigurnost podataka

No postoje i drugi, manje očiti problemi. Već prije je bilo spomenuto kako cloud poslužitelji koriste RAID konfiguraciju za osiguravanje integriteta podataka jer, nije pitanje hoće li se individualan disk pokvariti, već kada će se pokvariti i trebati zamjenu. Postavlja se pitanje: kad jedan od tih diskova bude isključen iz rada, gdje on završava? Ako poslužitelji koriste dobre prakse, redovno će provjeravati SMART status diskova i pravovremeno zamjeniti disk, a to znači da će taj disk i dalje funkcionira (no vjerojatno neće zadugo). Ako taj disk nije na odgovarajući način saniran, postoji mogućnost u kojoj neki zaposlenik osobno preuzme taj disk i podatke na njemu. To može rezultirati u komprimiranim SSH ključevima i širokom spektru osobnih korisničkih podataka.

Jedan primjer takve krađe podataka objavio je Bloomberg [10], prema kojem su 2 Shopify zaposlenika ukrali podatke korisnika. Podaci su uključivali e-mail adrese, imena, detalje narudžba i slično. U ovom primjeru su zaposlenici imali izravan pristup sustavu koji ima te informacije pa ih je bilo i jednostavnije uloviti na temelju zapisnika pristupa (*eng. log*), no u

slučaju odbačenih diskova koji i dalje funkcioniraju, veće su vjerojatnosti da takva krađa prođe nezamijećena budući da se na njih gleda kao otpad.

U idealnom svijetu, cloud poslužitelji bi vršili enkripciju svih svojih diskova, no to smanjuje performanse sustava pa se u praksi enkripcija ne implementira, no i dalje je moguća od strane korisnika (prikazano u sekciji 4.2). Enkripcija diskova od strane korisnika osigurala bi podatke u prije navedenom scenariju odbacivanje diskova, no i dalje ne osigurava apsolutnu privatnost podataka od samog cloud poslužitelje. Naime, čak i ako je cijeli disk enkriptiran, i dalje je potrebno da dio koji čeka SSH vezu za otključavanje diska bude izravno pohranjen na disku. Cloud poslužitelj mogao bi jednostavno izmijeniti izvršne datoteke i tako presresti sav korisnikov unos. Alternativno, moguće je izvršiti ispis memorije (*eng. memory dump*) u kojoj se nalazi ključ za enkripciju/dekripciju što nije praktično i jednostavno, ali je moguće [11].

3.1.3. Neizvjesnost

Ovo ovisi o poslužitelju, no generalno ne postoje garancije kod Cloud hostinga. Poslužitelj u bilo kojem momentu može odlučiti prestati nuditi usluge i ugasiti račune, a s tim i sve podatke. Ovakva situacija desila se 2019. godine kad su se 20 VPS poslužitelja odlučili povući s tržišta [12]. Nisu ponudili jasno objašnjenje i korisnicima su omogućili samo 2 dana za preuzimanje podataka prije nego formatiraju sve diskove. U slučaju da je neki korisnik promašio obavijest o isključenju, isti bi izgubio sve podatake. Zbog ovoga su sigurnosne kopije (*eng. backup*) od iznimne važnosti i nešto što se redovno treba prakticirati.

Tablica 1: VPS poslužitelji koji su 2019. najavili nagli prestanak rada

ArkaHosting	Bigfoot Servers	DCNHost	HostBRZ
HostedSimply	Hosting73	KudoHosting	LQHosting
MegaZoneHosting	n3Servers	ServerStrong	SnowVPS
SparkVPS	StrongHosting	SuperbVPS	SupremeVPS
TCNHosting	UMaxHosting	WelcomeHosting	X4Servers

3.2. Vlastito sklopovlje

Korištenje vlastitog sklopovlja nudi značajno veću razinu kontrole, no ne dolazi bez naplate. Početna cijena može biti značajno veća u odnosu na VPS, no potencijalno se može isplatiti kroz dulji vremenski period.

Manjak iskustva u sferi postavljanja servera može završiti u nesigurnoj i nepouzdanoj konfiguraciji. Npr. manjak UPS uređaja može stvarati česte zastoje (*eng. downtime*) u radu. Manjak rutinskog testiranja radne memorije može dovesti do različitih grešaka tijekom rada sustava čiji uzrok može biti teško otkriti. Manjak monitoringa SMART stanja diskova može dovesti do potpunog gubitka podataka u najgorem slučaju, a korupcije dijelova podataka u drugom.

3.2.1. Intel ME

Intel ME (*eng. Management Engine*) je podsustav svakog Intel procesora koji ima potpunu kontrolu nad sklopovljem. Ponekad se koristi za udaljenu administraciju uređaja u nekim poduzećima. Problem s Intel ME je da je to sustav bez granica koji bi, jednom kompromitiran, mogao zaobići sve sigurnosne mjere operacijskog sustava [13]. Zbog zatvorene prirode ove komponente, najbolje bi ju bilo onemogućiti, no to generalno nije moguće. Ako se za server koristi vlastito sklopovlje s Intel procesorom, moguće je *neutralizirati* Intel ME, odnosno, ukloniti većinu koda i time smanjiti ili u potpunosti eliminirati mogućnost značajne interakcije sa sustavom.

Intel ME Cleaner [14] skripta nudi mogućnost neutralizacije tog podsustava. Ovaj proces ovisi o sklopovlju, no generalni koraci su:

1. Izvršiti sigurnosnu kopiju BIOS slike
2. Primjeniti skriptu nad slikom
3. Instalirati izmijenjenu sliku natrag u BIOS

Nekad je moguće ovaj proces izvršiti u potpunosti programski, a nekad je potrebno koristiti vanjski SPI programer i izmjene raditi izravno na samom čipu što je van sklopa ovog rada.

3.3. Distribucija

Odabir distribucije uvelike ovisi o potrebama korisnika. U smislu politike nadogradnje, distribucije se dijele na 2 tipa:

- **LTS** - distribucija periodično zamrzava verzije programa, a sve nadogradnje su zapravo sigurnosne zakrpe ili popravci problema. Najpopularniji tip za servere jer nudi isto, predvidivo okruženje na neki dulji vremenski period.
- **Rolling** - distribucija nudi nove verzije programa kod svake nadogradnje. Nije popularno na serverima zbog manjka stabilnosti, no može biti korisno na osobnim računalima.

Iako LTS tip ima najviše smisla za server, sigurnost istog je potencijalno manja u odnosu na Rolling tip distribucije. LTS distribucije iz novih verzija Linux jezgre uzimaju samo sigurnosne zakrpe, dok ostale dijelove ne diraju. Problem kod ovog pristupa je mogućnost promašaja sigurnosnih zakrpa ako ih autori nisu jasno naznačili kao iste. Linus Torvalds [15] navađa kako često sigurnosne zakrpe na naznačuje kao takve jer bi time značajno olakšao napadačima otkrivanje i eksploatiranje ranjivosti.

Rolling model distribucije opet ima svoj set problema. Možda ovaj model pravovremeno dobiva sve sigurnosne zakrpe, ali dobiva i nove značajke što znači da potencijalno dobiva i nove greške koje mogu biti eksploatirane. Neke distribucije nude svojevrsni hibridni model u

kojem se može prilikom nadogradnje preuzimati uvijek nova verzija Linux jezgre, dok ostali programi koriste stabilne verzije. Jedna takva distribucija je Alpine Linux [16] koja nudi LTS i Edge repozitorij.

Zadana instalacija Alpine distribucije koristi LTS repozitorij koji je konfiguriran u `/etc/apk/repositories` datoteci:

```
# cat /etc/apk/repositories
https://dl-cdn.alpinelinux.org/alpine/v3.18/main
```

Naziv paketa LTS Linux jezgre je *linux-lts*. Moguće je koristiti rolling verziju jezgre bez mijenjanja repozitorija:

```
# apk add linux-edge
(1/1) Installing linux-edge (6.4.10-r0)
Executing busybox-1.36.1-r5.trigger
Executing kmod-30-r4.trigger
Executing mkinitfs-3.8.1-r0.trigger
==> initramfs: creating /boot/initramfs-edge
Executing grub-2.06-r13.trigger
Generating grub configuration file ...
...
Found linux image: /boot/vmlinuz-lts
Found initrd image: /boot/initramfs-lts
Found linux image: /boot/vmlinuz-edge
Found initrd image: /boot/initramfs-edge
done
...
```

Sada se koristi rolling verzija Linux jezgre, dok sve ostalo koristi LTS model. Uglavnom, odabir najboljeg modela distribucije je individual i ovisi o potrebama korisnika. Iako LTS model možda promaši koju zakrpu, generalno je dobar kompromis između sigurnosti i stabilnosti.

4. Lokalna sigurnost podataka

Ova sekcija odnosi se na redundantnost i sigurnost podatka na diskovima. Redundantnost se osigurava RAID konfiguracijom, dok se sigurnost osigurava enkripcijom.

4.1. SWAP

U većini situacija, potrebno je konfigurirati SWAP particiju ili datoteku koju će sustav koristiti u slučaju da nema dovoljno dostupne radne memorije. Radna memorija nije trajna, već se automatski briše prilikom gubitka napajanja budući da se tehnologija bazira na kondenzatorima. Ta značajka čini radnu memoriju idealnim mjestom za privremeno pohranjivanje senzitivnih informacija (u slučaju enkripcije diskova, to je ključ za kriptiranje i dekriptiranje), no ova sigurnosna značajka pada u vodu ako sustav krene zapisivati elemente radne memorije u SWAP (na disk).

Postoje 3 pristupa za rješavanje ovog problema:

1. Onemogućavanje SWAP prostora
2. Enkriptiranje SWAP prostora
3. ZRAM

Onemogućavanje SWAP prostora nije preporučljivo na računalima s malo radne memorije. Trajno onemogućavanje SWAP prostora vrši se uklanjanjem linija koje sadržavaju *swap* kao tip datotečnog sustava u `/etc/fstab` datoteci. Privremeno onemogućavanje SWAP particije do idućeg pokretanja računala moguće je sa sljedećom naredbom:

```
# swapoff -a
```

Enkriptiranje SWAP prostora moguće je na više načina, no jedan od jednostavnijih je koristiti SWAP datoteku unutar enkriptirane particije ili datoteke što je opisano u sekciji 4.2.

ZRAM je alternativa SWAP particiji koja radi na principu kompresiranja podataka prije nego oni budu zapisani u memoriju. Ovo može dovesti do ponešto nižih performansa od izravnog pristupa memoriji budući da se kontinuirano vrši kompresija i dekompresija. Unatoč tome, ZRAM će i dalje biti brži nego SWAP na disku. ZRAM se može jednostavno omogućiti koristeći pomoćne programe. Na Debian distribuciji, nudi se systemd-zram-generator [17]. Distribucije koje ne koriste SystemD obično nude zram-init [18] ili zramen [19]. Jednom instaliran, potrebno je omogućiti servis paketa, a procedura ovisi o init programu kojeg distribucija koristi.

4.2. Enkripcija

Enkripcija podataka moguća je na razini diska i na razini direktorija. Enkripcija na razini diska generalno je poželjnija budući da su svi podaci osigurani, a dodatno se ne vidi raspored

direktorija po disku ili tip datotečnog sustava što dodatno otežava napade. Enkripcija na razini direktorija je poželjnija kad su važne performanse budući da sistemske datoteke uglavnom nisu neka tajna, a dodatno omogućava korištenje odvojenih lozinka za različite datoteke.

U ovoj sekciji demonstrirana je enkripcija od strane operacijskog sustava, no neki diskovi nude i vlastiti mehanizam enkripcije. Jedan takav disk kojeg sastavlja Western Digital je "My Passport". Radi se o prijenosnom čvrstom disku koji ima mogućnost samo-enkripcije. Prema istraživanju [20], otkriveno je da je enkripcija veoma loše implementirana: moguće je zaobići autentifikaciju, ne koristi se kriptografski sigurna funkcija za generiranje slučajnih brojeva, nema provjere autentičnosti integriranih drivera i još više. Stoga je preporučljivo izbjegavati ovakav tip enkripcije te radije koristiti rješenje od strane operacijskog sustava koje je bilo dobro provjereno.

Linux desktop instalacije koje nude automatsko postavljanje enkripcije trenutno koriste kombinaciju LVM (*eng. Logical Volume Management*) datotečnog rasporeda unutar LUKS (*eng. Linux Unified Keyring System*) bloka.

Mobilni uređaji koriste enkripciju na razini direktorija. Android enkriptira samo korisnikove podatke. Enkripcija je esencijalna na mobilnim uređajima i čini preprodaju mogućom jer je dovoljno resetirati uređaj na tvorničke postavke pri čemu se brišu svi sažeti ključevi (*eng. hashed*) za enkripciju, zajedno s ostalim potrebnim informacijama u zaglavlju (*eng. header*) što čini vraćanje podataka nemogućim.

4.2.1. Na razini diska

Na razini diska koristi se LUKS standard, a interakcija s istim moguća je koristeći *cryptsetup* naredbu. Kako bi enkriptirali neki disk ili particiju, koristimo *luksFormat* argument, a neke od ključnih zastavica su:

- **–cipher** - algoritam za enkripciju. Popis dostupnih algoritama može se vidjeti u `/proc/crypto` datoteci. Zadana vrijednost je generalno *aes-xts-plain64*. Algoritam ovisi o potrebama korisnika, no treba izbjegavati *CBC* mod zbog sigurnosnih propusta [21].
- **–hash** - funckija sažimanja korištena kod derivacije ključa. Zadana vrijednost je sha256.
- **–key-size** - veličina ključa u bitovima. XTS mod enkripcije koristi polovicu veličine ključa pa za 128 bita treba unijeti 256. Zadana vrijednost za aes je 256 bitova, no 128 može biti poželjnija na sporijem sklopovlju budući da je ta varijacija brža [22].
- **–sector-size** - veličina sektora. Zadana vrijednost je 512 bajta, no korištenje 4096 bajta je poželjnije jer su performanse značajno bolje [23].

S *lsblk* naredbom možemo vidjeti trenutne diskove:

```
# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
vda    253:0    0    10G  0 disk
vda1 253:1    0   500M  0 part /boot
```

```
vda2 253:2    0   4.5G  0 part /
vda3 253:3    0   2.5G  0 part
vda4 253:4    0   2.5G  0 part
```

Za particiju *vda3* biramo aes-xts-plain64, sha256 i 128 bitova za veličinu ključa:

```
# cryptsetup luksFormat /dev/vda3 --cipher aes-xts-plain64 \
                                  --hash sha256 --key-size 256
WARNING!
========
This will overwrite data on /dev/vda3 irrevocably.
Are you sure? (Type 'yes' in capital letters): YES
Enter passphrase for /dev/vda3: ***
Verify passphrase: ***
```

Disk otključavamo sa navedenom naredbom, gdje je *DISK* naziv dekriptiranog prostora:

```
# cryptsetup open /dev/vda3 DISK
Enter passphrase for /dev/vda3: ***
# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
vda      253:0    0    10G  0 disk
vda1   253:1    0   500M  0 part  /boot
vda2   253:2    0   4.5G  0 part  /
vda3   253:3    0   2.5G  0 part
 DISK 252:0    0   2.5G  0 crypt
vda4   253:4    0   2.5G  0 part
```

Otključani prostor diska sada je dostupan pod `/dev/mapper/DISK`. Taj prostor moguće je formatirati s proizvoljnim datotečnim sustavom od kojeg je enkripcija apstrahiana. Disku je moguće dodati datoteku kao ključ putem koje je onda moguće raditi automatiziranu dekripciju prilikom pokretanja računala. Možemo koristiti *dd* naredbu za generaciju datoteke proizvoljne veličine:

```
# dd if=/dev/urandom of=key count=128
128+0 records in
128+0 records out
65536 bytes (66 kB, 64 KiB) copied, 0.00047326 s, 138 MB/s
# cryptsetup luksAddKey /dev/vda3 key
Enter any existing passphrase: ***
```

Automatsko otključavanje particije generalno je moguće preko `/etc/crypttab` datoteke. Ovaj pristup koristi se kad particija u pitanju nije ona na kojoj je sustav instaliran. Ključ je bio generiran u `/root` direktoriju. Iako smo se na uređaj referirali sa `/dev/vda3`, taj naziv nije nužno statičan i može se promijeniti. Potrebno je koristiti unikantan identifikator, a jedan takav je UUID. UUID se može dohvatiti preko *lsblk -f* naredbe. U odnosu na navedeno, crypttab linija izgledala bi ovako:

```
DISK        UUID=5c9293f1-eb46-4a4c-b4d0-232a6e97024c       /root/key
```

Otključavanje preko crypttab datoteke događa se tek nakon što je sustav inicijaliziran. Kako bi diskove otključali ranije (potrebno kad je sam sustav enkriptiran), potrebno je dodati pripadne argumente Linux jezgri (*eng. cmdline*). Enkripcija sustava vrši se prije instalacije i uvelike ovisi o načinu instalacije (rootfs instalacije u odnosu na automatizirani instalacijski program), a i o samoj distribuciji (ne koriste sve distribucije isti program za generaciju initramfs prostora, a u konfiguraciji istog potrebno je specificirati potrebne module za enkripciju, datotečne prostore i slično).

S obzirom na navedeno, u ovom radu se neće vršiti cijela instalacija, no navedene su neke generalne smjernice. Generalno, standardni *bootloader* jer GRUB, a za generaciju *initramfs* datotečnog sustava uglavnom se koristi *dracut*. U datoteci `/etc/default/grub` možemo specificirati dodatne argumente za Linux jezgru. Polje za specificiranje dodanih argumenata je `GRUB_CMDLINE_LINUX_DEFAULT`.

Razmotrimo primjer u kojem imamo btrfs u RAID1 konfiguraciji s 2 diska koja su individualno enkriptirani s cryptsetup. Sustav je instaliran na navedenoj enkriptiranoj RAID1 konfiguraciji (nad ROOTp1 i ROOTp2 dekriptiranim prostorima). Dekriptirana `/boot` particija nalazi se na `/dev/loop0p1` particiji.

```
# lsblk -f
NAME        FSTYPE      FSVER LABEL     UUID
loop0
  loop0p1
  loop0p2   crypto_LUKS                 a96055ea-62fc-40f2-9867-4dfb14cc16c9
   KLJUC
  loop0p3   crypto_LUKS                 a2cd4901-b673-4bc3-b0c4-bfb68ec52769
    ROOTp1
loop1       crypto_LUKS                 e8618b66-9c43-4fc3-a366-ca32325b445d
  ROOTp2
```

Kako bi dekripcija bila moguća koristeći *dracut* initramfs i GRUB bootloader, potrebno je specificirati sljedeće parametre u `/etc/default/grub`:

```
root=LABEL=ROOT
rd.luks.name=a2cd4901-b673-4bc3-b0c4-bfb68ec52769=ROOTp1
rd.luks.name=e8618b66-9c43-4fc3-a366-ca32325b445d=ROOTp2
rd.luks.uuid=keysource:a96055ea-62fc-40f2-9867-4dfb14cc16c9
rd.luks.key=/kljuc:LABEL=KLJUC
```

S *root* definiramo root datotečni sustav. Btrfs datotečni sustav bio je formatiran s imenom *ROOT* pa koristimo `LABEL` deklaraciju za identifikaciju datotečnog sustava. S `rd.luks.name` definiramo UUID enkriptiranih diskova zajedno s imenom koje želimo nadjenuti dekriptiranom prostoru. Enkriptirani disk na kojem se nalazi ključ kojim se mogu dekriptirati oba diska definiramo s `rd.luks.uuid=keysource:UUID`, a s `rd.luks.key=/kljuc:LABEL=KLJUC`

definiramo putanju na dekriptiranom disku gdje je taj ključ dostupan, zajedno s imenom datotečnog sustava kojeg koristi. Prilikom podizanja sustava, tražit će se lozinka da dekripciju `/dev/loop0p2` diska. Ako unesemo točnu lozinku, disk će se dekriptirati, a ključ kojeg dekriptirani prostor sadržava koristiti će se za automatsku dekripciju `/dev/loop0p3` i `/dev/loop1` diska.

4.2.2. Na razini datotečnog sustava

Za ovaj način enkripcije, koristi se *fscrypt* alat, a isti podržava ext4, F2FS i UBIFS datotečne sustave [24]. Ovdje će se koristiti već prije formatirana ext4 particija `/dev/vda4` dostupna na putanji `/opt`.

Prvo je potrebno omogućiti enkripciju u postavkama datotečnog sustava:

```
# tune2fs -O encrypt /dev/vda4
tune2fs 1.47.0 (5-Feb-2023)
```

Zatim inicijaliziramo fscrypt (globalne postavke i za putanju):

```
# fscrypt setup
Defaulting to policy_version 2 because kernel supports it.
Customizing passphrase hashing difficulty for this system...
Created global config file at "/etc/fscrypt.conf".
...
# fscrypt setup /opt
...
```

Stvaramo novi direktorij te omogućavamo enkripciju:

```
# mkdir /opt/kriptirano
# fscrypt encrypt /opt/kriptirano
The following protector sources are available:
1 - Your login passphrase (pam_passphrase)
2 - A custom passphrase (custom_passphrase)
3 - A raw 256-bit key (raw_key)
Enter the source number for the new protector [2 - custom_passphrase]: 2
Enter a name for the new protector: protector
Enter custom passphrase for protector "protector": ***
Confirm passphrase: ***
"kriptirano" is now encrypted, unlocked, and ready for use.
```

Stvaramo novu datoteku s tajnim sadržajem, zaključa damo direktorij i vidimo da su sadržaj i ime datoteke enkriptirani:

```
# echo sadrzaj > /opt/kriptirano/tajna.txt
# fscrypt lock /opt/kriptirano
"/opt/kriptirano/" is now locked.
# ls /opt/kriptirano
VEaPZs2JNzZ0Hj_uaX0swJfh_XmAKzbqVJ5tSCx9pcG_I6UhxUd4_A
# cat /opt/kriptirano/VEaPZs2JNzZ0Hj_uaX0swJfh_XmAKzbqVJ5tSCx9pcG_I6UhxUd4_A
cat: can't open 'enc/VEaPZs2JNzZ0Hj_uaX0swJfh_XmAKzbqVJ5tSCx9pcG_I6UhxUd4_A':
    Required key not available
```

Fscrypt je koristan kod višekorisničkih sustava gdje svaki korisnik može imati vlastiti ključ za enkripciju, dok ostatak sustava nije tajna. Dodatno, odvojeni direktoriji mogu imati druge lozinke što umanjuje štetu ako jedan od ključeva kompromitiran.

4.2.3. Na razini direktorija

Enkripcija na razini direktorija manje je standardizirana pa postoji više različitih opcija. Pozitivna strana ovog pristupa je da enkripcija radi na bilo kojem datotečnom sustavu. Npr. fscrypt se ne može koristiti u slučaju da se želi enkriptirani samo jedan direktoriji unutar *btrfs* datotečnog sustava, no ovo su neki od alata koji to omogućavaju:

- gocryptfs
- cryptomator
- cryfs
- encfs

Ovdje će se ukratko demonstrirati gocryptfs. Stvaramo 2 proizvoljna direktorija: *enkriptirano* i *dekriptirano*. Inicijaliziramo gocryptfs:

```
# mkdir enkriptirano dekriptirano
# gocryptfs -init enkriptirano
Choose a password for protecting your files.
Password: ***
Repeat: ***
...
```

Zatim otključavamo enkriptirani direktorij i transparento dekriptirani sadržaj montiramo u *dekriptirano* direktorij:

```
# gocryptfs  enkriptirano dekriptirano
Password: ***
Decrypting master key
Filesystem mounted and ready.
```

Sve daljne operacije vršimo u *dekriptirano* direktoriju. Svi podaci se automatski enkriptiraju i spremaju u *enkriptirano* direktorij.

```
# touch dekriptirano/tajna
# umount dekriptirano
# ls dekriptirano

# ls enkriptirano
gocryptfs.conf   gocryptfs.diriv   r2LVc4f2e6hYPaAtPNbn6Q
```

4.3. RAID

RAID (*eng. Redundant Array of Inexpensive Disks*) često se koristi za osiguravanje postojanja kopije podataka, ali i za poboljšanje performansa. Svaki pohrambeni medij ima svoj životni vijek kojeg je teško predvidjeti (moguće je provjeravati stanje medija koristeći *smartctl* ili neki drugi alat, ako disk podržava SMART) pa se u praksi koristi neka razina redundancije. Postoji softverski i hardverski RAID. Ovdje se razmatra softverski RAID budući da je fleksibilniji i omogućava korištenje bilo kojeg diska. Postoje različiti tipovi RAID konfiguracije, no ovdje će se samo razmatrati RAID1. RAID1 je jednostavan: on zrcali sadržaj jednog diska na drugi. Ova konfiguracija zahtjeva minimalno 2 diska. U slučaju postojanja samo 2 diska, oba se moraju pokvariti za potpuni gubitak podataka. Dodatno, RAID1 poboljšava performanse datotečnog sustava budući da se dijelovi tražene datoteke mogu dohvaćati paralenog s 2 diska. U ovoj sekciji je demonstrirana klasična RAID konfiguracija s *mdadm*, no prikazana je i alternativna verzija na razini datotečnog sustava (*btrfs*).

Važno je napomenuti da bilo koja RAID konfiguracija nije ekvivalentna sigurnosnoj kopiji (*eng. backup*). Uvijek postoji mogućnost kvara više diskova u isto vrijeme što bi dovelo do gubitka podataka.

4.3.1. mdraid

Mdadm [25] je alat za softverski RAID službeno podržan od strane Linux jezgre. Ovo je najčešći način konfiguracije RAID-a unutar Linux sustava. RAID1 ovdje vrši potpuno zrcaljenje na 2 diska. Dakle, datotečni sustav koji će se kasnije stvarati nad `/dev/md0` RAID virtualnim uređajem će se zapravo paralelno stvarati na oba diska, samo što je to apstrahirano od ostatka sustava. Iako ovaj pristup nudi zaštitu podatka u slučaju kvara jednog diska, on ne vrši nikakvo zacjeljivanje (*eng. self-healing*) u slučaju korupcije podataka (*eng. bitrot*) na jednom disku.

```
# lsblk
NAME   MAJ:MIN RM  SIZE RO TYPE MOUNTPOINTS
vda      253:0    0    10G  0 disk
vda1   253:1    0   500M  0 part  /boot
vda2   253:2    0   4.5G  0 part  /
vda3   253:3    0   2.5G  0 part
vda4   253:4    0   2.5G  0 part
```

Koristiti će se particije `/dev/vda3` i `/dev/vda4` za stvaranje RAID1 konfiguracije. Budući da ove particije nemaju *boot* zastavicu koja je potrebna u slučaju da pokrećemo sustav s RAID1 prostora, mdadm traži potvrdu da ne očekujemo isto.

```
# mdadm --create --verbose /dev/md0 --level=1 --raid-devices=2 /dev/vda3 /dev/vda4
mdadm: Note: this array has metadata at the start and
    may not be suitable as a boot device.  If you plan to
    store '/boot' on this device please ensure that
    your boot-loader understands md/v1.x metadata, or use
    --metadata=0.90
mdadm: size set to 2618368K
Continue creating array? (y/n) y
mdadm: Defaulting to version 1.2 metadata
mdadm: array /dev/md0 started.
```

Sada formatiramo RAID1 prostor s ext4 datotečnim sustavom, spremamo raid konfiguraciju kako bi bila automatski dostupna kod sljedećeg pokretanja računala, montiramo novostvoreni datotečni sustav i opet spremamo konfiguraciju za njega u `/etc/fstab` da bi se on automatski montirao kod pokretanja računala.

```
# mkfs.ext4 -L RAID1 /dev/md0
mke2fs 1.47.0 (5-Feb-2023)
...
# mdadm --detail --scan >> /etc/mdadm.conf
# mount -t ext4 /dev/md0 /mnt
# grep /dev/md0 /etc/mtab >> /etc/fstab
```

Općenito, stanje RAID uređaja možemo vidjeti u `/proc/mdstat` datoteci:

```
# cat /proc/mdstat
Personalities : [raid1]
md0 : active raid1 vda4[1] vda3[0]
      2618368 blocks super 1.2 [2/2] [UU]

unused devices: <none>
```

RAID konfiguraciju jednostavno je isključiti:

```
# mdadm --stop /dev/md0
```

4.3.2. btrfs

Btrfs je datotečni sustav koji koristi COW (*eng. Copy on write*) pristup kod pisanja podataka za razliku od ostalih datotečnih sustava koji generalno izravno pišu promjene. Uz to, btrfs zapisuje kontrolni zbroj (*eng. checksum*) blokova što omogućava detektiranje grešaka. Dodatno, btrfs omogućava RAID konfiguraciju, no ona funkcionira drugačije od mdraid. Btrfs

datotečni sustav može koristiti više različitih diskova, a u RAID1 konfiguraciji ne vrši zrcaljenje, već osigurava da dan podatak u bilo kojem trenutku bude prisutan na 2 diska. U slučaju prisutnosti 3 diska, RAID1 bi isti podatak postavio na samo 2 od 3 diska, dok bi mdraid taj podatak postavio na sva 3.

Particije `/dev/vda3` i `/dev/vda4` formatiramo s btrfs datotečnim sustavom s oznakom *RAID1*. Zastavica **-m** koristi se za postavljanje RAID profila za metapodatke, dok **-d** za podatke. Ovdje koristimo RAID1 profil za oba parametra, ali je inače moguće koristiti različite profile.

```
# mkfs.btrfs -L RAID1 -mraid1 -draid1 /dev/vda3 /dev/vda4
btrfs-progs v6.3.2
See https://btrfs.readthedocs.io for more information.
...
Block group profiles:
  Data:             RAID1           256.50MiB
  Metadata:         RAID1           256.00MiB
  System:           RAID1             8.00MiB
..
Checksum:           crc32c
Number of devices:  2
...
```

Zatim montiramo disk. Moguće je specificirati dodatne opcije s **-o** zastavicom. Ovdje omogućavamo transparentu kompresiju koristeći zstd algoritam te onemogućavamo spremanje vremena pristupa datotekama:

```
# mount -o compress-force=zstd,noatime -L RAID1 /mnt
```

Btrfs RAID1 sigurniji je od mdraid verzije zbog kontrolnog zbroja koji omogućava ispravak koruptiranih podataka. Podaci se prilikom čitanja automatski [26] popravljaju s redundantnom kopijom, no moguće je izvršiti ručnu provjeru cijelog diska koristeći *scrub naredbu*. Status provjere dostupan je sa *status* argumentom:

```
# btrfs scrub start /mnt
scrub started on /mnt, fsid 86375923-a903-4c14-951f-2201e242459e (pid=5010)
# btrfs scrub status /mnt
...
Status:           finished
Duration:         0:00:00
Total to scrub:   288.00KiB
Rate:             0.00B/s
Error summary:    no errors found
```

Veoma korisna značajka btrfs-a je mogućnost stvaranja snimke (*eng. snapshot*) trenutnog stanja datotečnog sustava ili samo njegovog dijela (*eng. subvolume*). Prvo stvaramo direktorij u kojem ćemo pohranjivati snimke, a iste se stvaraju ovako:

```
# mkdir /mnt/@snapshots
# btrfs subvolume snapshot -r /mnt /mnt/@snapshots/backup1
Create a readonly snapshot of '/mnt' in '/mnt/@snapshots/backup1'
```

Ovime smo stvorili snimku trenutnog stanja datotečnog sustava koju uvijek možemo pregledati navigirajući se u direktorij `/mnt/@snapshots/backup1`. Sigurnosne kopije kod btrfs-a moguće je preuzeti preko mreže koristeći *ssh*. S udaljenog računala možemo preuzeti snimku na lokalni btrfs datotečni sustav. S lokalnog računala pozivamo btrfs *send* naredbu na udaljenom poslužitelju za preuzimanje snimke preko sigurne veze, a nju preusmjeravamo u lokalnu btrfs *receive* naredbu s btrfs putanjom na kojoj želimo spremati snimke poslužitelja:

```
# ssh root@domena 'btrfs send /mnt/@snapshots/backup1' | btrfs receive /mnt/@server
    -backups
```

Nakon preuzimanja prve snimke, na poslužitelju nakon nekog vremena stvaramo drugu na putanji `/mnt/@snapshots/backup2`. Sada nije potrebno ponovno preuzimati cijelu snimku, već možemo preuzeti samo promjene u odnosu na prvu snimku:

```
# ssh root@domena 'btrfs send -p /mnt/@snapshots/backup1 /mnt/@snapshots/backup2' |
     btrfs receive /mnt/@server-backups
```

Ovo je efikasna backup strategija koja se može automatizirati, no s vremenom može postati komplicirana pa se mogu koristiti pomoćni programi za btrfs sigurnosne kopije kao što je npr. btrbk [27].

5. Docker

Docker je alat za kontejnerizaciju programa čime se osigurava veća sigurnost i reproducibilnost u odnosu na izravnu instalaciju. Docker kontejneri koriste odvojeno okruženje od operacijskog sustava na kojem se nalaze, no dijele istu jezgru što dovodi do boljih performansa i manjeg utroška resursa u odnosu na prave virtualne mašine.

Docker kontejner može se stvoriti i pokrenuti koristeći *docker run* naredbu, no u praksi se koristi *docker-compose.yaml* datoteka za jednostavniju orkestraciju kontejnera.

5.1. Korisnički račun

Osim ako je drugačije specificirano, svaki Docker kontejner pokreće se kao *root* korisnik, no i dalje je ograničen u svojim mogućnostima. Ako zlonamjerni program s root privilegijama unutar kontejnera uspije izaći iz istog na temelju neke ranjivosti u Docker-u, tada on ima i root van tog kontejnera [28]. Ovo se može prevenirati izbjegavanjem korištenja root računa unutar kontejnera.

Ne korištenje root računa unutar Docker kontejnera nekad je problematično. Razmatrajmo sljedeću *docker run* naredbu za pokretanje Postgresql aplikacije kao korisnik s identifikacijskim brojem (*eng. UID*) 1000:

```
# mkdir /opt/postgresql
# chown 1000:1000 /opt/postgresql
# docker run --rm -it
             -v /opt/postgresql:/lib/postgresql/data
             -e POSTGRES_PASSWORD=123
             --user 1000:1000
             postgres:15.3-alpine
chmod: /var/lib/postgresql/data: Operation not permitted
chmod: /var/run/postgresql: Operation not permitted
The files belonging to this database system will be owned by user "postgres".
This user must also own the server process.
...
fixing permissions on existing directory /var/lib/postgresql/data ... initdb: error:
     could not change permissions of directory "/var/lib/postgresql/data": Operation
     not permitted
```

Postgresql daje grešku prilikom inicijalizacije. Problem leži u činjenici da *initdb* komponenta Postgresql kontejnera zahtjeva root privilegije, no ne i sam Postgresql [29]. Ovo se rješava pokretanjem kontejnera kao root korisnik, nakon čega se prilagodi vlasništvo datoteka koje je Postgresql generirao i, na kraju, opet pokrene, ali kao korisnik s UID 1000:

```
# docker run --rm -it
             -v /opt/postgresql:/lib/postgresql/data
             -e POSTGRES_PASSWORD=123
```

```
            postgres:15.3-alpine
...
PostgreSQL init process complete; ready for start up.
...
database system is ready to accept connections
# ^C
# chown -R 1000:1000 /opt/postgresql
# docker run --rm -it
            -v /opt/postgresql:/lib/postgresql/data
            -e POSTGRES_PASSWORD=123
            --user 1000:1000
            postgres:15.3-alpine
...
database system is ready to accept connections
```

Ovo je primjer za Postgresql, no postoje i mnogi drugi takvi kontejneri gdje su potrebni razni međukoraci za pokretanje bez root privilegija. Postoje i kontejneri koji pojednostavljuju ovaj proces na način da kontejner ima ugrađen minimalan init sistem kao što je *s6*. Ovaj pristup funkcionira na način da init program unutar kontejnera ima root privilegije, no on onda pokreće samu aplikaciju kontejnera bez root privilegija. Popularan distributer takvih slika je linuxserver.io [30] koji koristi *s6-overlay* za init sistem, a on opet koristi ID korisnika definiran preko okolišne (*eng. environmental*) varijable.

Sljedeći primjer koristi linuxserver.io sliku za ddclient te definira korištenje korisnika s UID 1000. S *docker top* naredbom možemo vidjeti da ddclient proces koristi UID 1000 iako sam kontejner ima root privilegije. Sve dok ne postoje neki sigurnosni problemi sa s6-overlay, ovaj način *rootless* korištenja kontejnera sličan je specificiranju **–user** zastavice kod *docker run* naredbe, no može dovesti do manjka efikasnosti kad se koristi više takvih slika budući da će u svakoj postojati ti dodani pomoćni procesi.

```
# docker run -d   \
  --name=ddclient \
  -e PUID=1000    \
  -e PGID=1000    \
  lscr.io/linuxserver/ddclient
# docker top ddclient
UID                 CMD
root                /package/admin/s6/command/s6-svscan -d4 -- /run/service
root                s6-supervise s6-linux-init-shutdownd
root                bash ./run svc-inotify
1000                ddclient - sleeping for 280 seconds
...
```

5.2. Capabilities

Neke aplikacije vrše radnje koje zahtjevaju specifične elevirane privilegije zbog kojih nema smisla cijelu takvu aplikaciju pokretati kao root, već se taj problem može adresirati do-

davanjem dodatnih mogućnosti toj izvršnoj datoteci. Na taj način aplikacija ima samo neke mogućnosti root računa.

Budući da je web server otvoren prema internetu, u najboljem je interesu korisnika ograničiti njegove mogućnosti što se vrši pokretanjem web servera kao običan korisnik, no to se na prvu ruku čini nemogućim. Naime, web serveri pružaju sadržaj korisnicima na portovima 80 (http) i 443 (https). Svi portovi ispod 1024 su privilegirani i stoga zahtjevaju elevirane root privilegije. Ovaj problem može se rješiti dodavanjem `NET_BIND_SERVICE` mogućnosti. U ovom je slučaju potrebna i `NET_RAW` mogućnost za korištenje RAW i PACKET utičnica (*eng. socket*).

U nastavku slijedi docker-compose.yaml primjer za Caddy [31] web server kojeg pokrećemo kao običan korisnik s **cap_add** deklaracijom, dok sve ostale mogućnosti onemogućavamo s **cap_drop** deklaracijom.

```
version: "3"
services:
  caddy:
    image: caddy:2.7.4-alpine
    container_name: Caddy
    restart: unless-stopped
    user: 1000:1000
    volumes:
      - ./app/Caddyfile:/etc/caddy/Caddyfile
      - ./app/config:/config
      - ./app/data:/data
    network_mode: "host"
    cap_drop:
      - ALL
    cap_add:
      - NET_BIND_SERVICE
      - NET_RAW
```

Kad ne bi dodali prije navedene mogućnosti, kontejner se ne bi mogao uspješno pokrenuti:

```
# docker logs Caddy
exec /usr/bin/caddy: operation not permitted
```

Općenito se preporuča koristiti *ALL* vrijednost za **cap_drop** direktivu budući da prema zadanim vrijednostima Docker daje različite mogućnosti kontejneru, neke od kojih su [32]:

- **CHOWN** - mijenjanje vlasništva datoteka
- **KILL** - slanje signala procesima
- **SETFCAP** - izmjena mogućnosti izvršne datoteke

5.3. Mrežni koncepti

Kod pokretanja Docker kontejnera bez dodatne konfiguracije, kontejner dobiva vlastitu mrežu odvojenu od sustava na kojoj je pokrenut te ne može komunicirati s drugim kontejnerima. Sljedeće sekcije demonstriraju različite načine komunikacije kontejnera. Za primjer se koristi Caddy web server koji djeluje kao *reverse proxy* za Gitea git server, a iz docker-compose.yaml datoteke su prikazane samo relevantne linije (nema konfiguracije volumena itd.).

5.3.1. Obostrana komunikacija s drugim kontejnerima

5.3.1.1. Koristeći isti docker-compose

Komunikacija dviju ili više kontejnera jednostavno se omogućava specificiranjem tih kontejnera u istoj docker-compose.yaml datoteci:

```
services:
  caddy:
    image: caddy:2.7.4-alpine
    container_name: Caddy
  gitea:
    image: gitea/gitea:1.20.2-rootless
    container_name: Gitea
```

Za adresu se koristi **container_name** direktiva. Demonstracija komunikacije: ulazimo u Caddy kontejner i vršimo *ping* za Gitea kontejner:

```
# docker exec -it Caddy ping -c 1 Gitea
PING Gitea (192.168.176.3): 56 data bytes
64 bytes from 192.168.176.3: seq=0 ttl=42 time=0.166 ms

--- Gitea ping statistics ---
1 packets transmitted, 1 packets received, 0\% packet loss
round-trip min/avg/max = 0.166/0.166/0.166 ms
```

5.3.1.2. Koristeći odvojeni docker-compose

Ponekad je poželjno dijeliti jedan kontejner s više drugih, a da se pritom održi izolacija kontejnera iz sigurnosnih razloga. Ovo je korisno kad više različitih kontejnera treba pristup nekoj bazi podataka. U slučaju većeg broja kontejnera, svaki bi trebo vlastitu instancu s vlastitom konfiguracijom što otežava održavanje, a i dovodi do veće potrošnje resursa. Ovdje razmatramo Postgresql bazu podataka koju će koristiti Gitea git server.

U docker-compose.yaml za Postgresql potrebno je deklarirati mreže u odvojenom **networks** bloku te Postgresql spojiti na iste koristeći **networks** deklaraciju. Na ovaj način činimo mreže *net_gitea* i *net_servis* dostupnima drugim docker-compose datotekama.

```
version: "3"
services:
  postgresql:
    image: postgres:15.3-alpine
    container_name: Postgresql
    environment:
      - POSTGRES_PASSWORD=123
    networks:
      - net_gitea
      - net_servis

networks:
  net_gitea:
    name: net_gitea
  net_servis:
    name: net_servis
```

Sada se navigiramo u drugi direktorij u kojem postavljamo docker-compose za Gitea kontejner. Opet definiramo mreže putem odvojenog **networks** bloka. Dodajemo mrežu s istim imenom kao u prethodnoj datoteci te postavljamo i **external** direktivu s vrijednošću *true*. Ovo govori Docker-u da ne stvara novu mrežu, već koristi postojeću. S direktivom **networks** spajamo Gitea kontejner na tu mrežu.

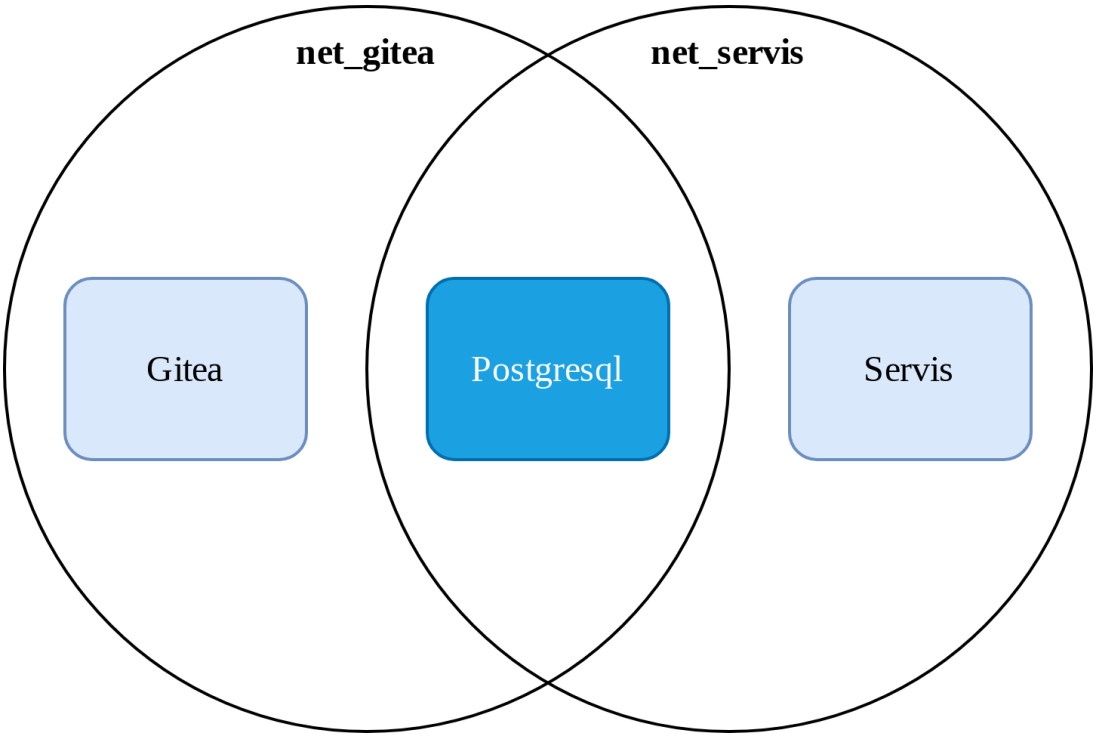
```
version: "3"
services:
  gitea:
    image: gitea/gitea:1.20.2-rootless
    container_name: Gitea
  networks:
    - net_gitea
networks:
  net_gitea:
    external: true
```

Pokušavamo pokrenuti Gitea kontejner:

```
# cd /opt/gitea
# docker-compose up -d
network net_gitea declared as external, but could not be found
```

Mreža ne postoji pa ju moramo stvoriti. Najjednostavniji način je prvo pokrenuti docker-compose za Postgresql, a zatim za Giteu:

```
# cd /opt/postgresql && docker-compose up -d
[+] Running 2/2
   Network net_gitea Created
   Network net_servis Created
   Container Postgresql Running
```



Slika 2: Odvojene mreže koje dijele isti kontejner

```
# cd /opt/gitea && docker-compose up -d
[+] Running 1/1
   Container Gitea Running
```

Sada je moguća izravna komunikacija između kontejnera. U primjeru je stvorena i mreža *net_ostalo*. Naime, moguće je koristiti mrežu *net_gitea* za sve druge kontejnere koji trebaju pristup Postgresql instanci, no onda bi svi međusobno bili dostupni jedni drugima. Kako bi minimizirali štetu u slučaju uspješnog napada na jedan od kontejnera, možemo definirati odvojenu mrežu za svaki kontejner u Postgresql docker-compose datoteci, a onda individualne mreže dodavati u pripadne docker-compose datoteke.

5.3.2. Dostupnost portova kontejnera operacijskom sustavu

Željene portove potrebno je navesti unutar **ports** direktive. Vrijednost porta s desne strane dvotočke odnosi se na port unutar kontejnera dok se lijeva odnosi na vanjski port kojeg sustav vidi i preko kojeg može pristupiti portu koji se nalazi unutar kontejnera. Gitea nudi svoj primarni servis na portu 3000 unutar kontejnera te ga čini dostupnim na lokalnoj mreži računala na portu 4000 što je osigurano specificiranjem adrese 127.0.0.1. Gitea kontejner također nudi *ssh* servis na portu 22 unutar kontejnera, a za razliku od prethodnog primjera, čini ga dostupnim ne samo lokalnom računalu, već cijelom svijetu na portu 2222.

```
services:
  caddy:
    image: caddy:2.7.4-alpine
    container_name: Caddy
```

```
  gitea:
    image: gitea/gitea:1.20.2-rootless
    container_name: Gitea
    ports:
      - "127.0.0.1:4000:3000"
      - "2222:22"
```

5.3.3. Dostupnost mreže operacijskog sustava kontejneru

Nekim je kontejnerima potreban izravan pristup mreži domaćina (*eng. host*) na kojoj Docker radi. Ovo je moguće postavljanjem **network_mode** direktive na host. U ovom primjeru Caddy kontejner ima izravan pristup mreži domaćina dok je Gitea ograničena na svoju mrežu. Izravna komunikacija korištenjem **container_name** kao adrese nije moguća, Caddy može komunicirati s Gitea kontejnerom koristeći portove koje ona otvara (4000 i 2222).

```
services:
  caddy:
    image: caddy:2.7.4-alpine
    container_name: Caddy
    user: 2000:2000
    network_mode: host
  gitea:
    image: gitea/gitea:1.20.2-rootless
    container_name: Gitea
    ports:
      - "127.0.0.1:4000:3000"
      - "2222:22"
```

Host način mreže treba izbjegavati budući su takvom kontejneru svi mrežni servisi domaćina dostupni. Jedan primjer gdje je koristan je s Eturnal [33] TURN serverom koji se koristi za VOIP razgovore. Eturnal serveru potreban je veći broj UDP portova što nije idealno za Docker koji ima podosta loše performanse kod mapiranja velikog broja portova [34] pa se stoga preferira *host* način mreže.

5.4. Ograničavanje resursa

Ograničavanje resursa poželjno je u slučaju većeg broja kontejnera kako ne bi došlo do zagušenja. Na ovaj se način mogu prioritetizirati kontejneri, a ograničenje dodatno štiti od mogućnosti utroška sve dostupne memorije što bi moglo dovesti do zamrzavanja sustava.

Neke od korisnih opcija za ograničavanje konzumpcije resursa kontejnera [35]:

RAM

- **–memory** - maksimalna alokacija memorije (najmanje 6MB)

- **–memory-swap** - ima učinak samo ako je *memory* opcija postavljena. Ukratko, predstavlja maksimalnu alokaciju memorije. U slučaju da je *memory* postavljen na 100MB, a *memoryswap* na 500MB, kontejner može koristiti 100MB memorije i 400MB SWAP-a.

CPU

- **–cpus** - decimalni broj koji naznačava koliko procesorskih jezgara kontejner može koristiti

5.5. Vatrozid

Na Linux sustavima, Docker koristi *iptables* za mrežna pravila kontejnera [36]. Iako je moguće koristiti druge programe za menadžment vatrozida, proces je često nejasan i zahtjeva značajne izmjene pa je najbolje vatrozid imat podešen s iptables kad se koristi docker.

Kad se želi ograničiti pristup nekom portu, tradicionalno se to radi nad `INPUT` lancem (*eng. chain*). Kad bi željeli ograničiti pristup TCP portu 22 na mrežnom adapteru *eth0*, to možemo učiniti s:

```
# iptables -A INPUT -i eth0 -p tcp --dport 22 -j DROP
```

Situacija je kompliciranija kad ovo pokušavamo s Docker-om. Naime, `INPUT` lanac nema utjecaj na Docker kontejnere. Docker stvara 2 iptables lanca:

- **DOCKER** - lanac s automatski generiranim pravilima za kontejnere koji se ne bi trebao ručno izmjenjivati
- **DOCKER-USER** - lanac koji se smije izmjenjivati i koji se primjenjuje prije `DOCKER` lanca. Ovdje se može ograničiti pristup portovima te ostale operacije.

Kad bi željeli prethodni primjer za port 22 primjeniti za Docker kontejnere, iptables naredba izgledala bi ovako:

```
# iptables -I DOCKER-USER -i eth0 -p tcp --dport 22 -j DROP
```

Koristi se **-I** umjesto **-A** jer je potrebno pravilo postaviti prije kraja lanca budući da kraj uključuje *RETURN* poziv koji je potreban za nastavak evaluacija iptables pravila koje se nalaze u `DOCKER` lancu:

```
 # iptables -L DOCKER-USER
Chain DOCKER-USER (1 references)
target     prot opt source               destination
DROP       tcp  --  anywhere             anywhere             tcp dpt:ssh
RETURN     all  --  anywhere             anywhere
```

Slijedi primjer docker-compose datoteke gdje bi ova pravila bila valjana:

```
  services:
  gitea:
    image: gitea/gitea:1.20.2-rootless
    container_name: Gitea
    ports:
      - "22:22"
```

Razmotrimo docker-compose datoteku u kojoj koristimo *host* način mreže za Caddy, dok za Gitea server mrežu kontejneriziramo. Caddy konfiguracija koristi portove 80 i 443.

```
services:
  caddy:
    image: caddy:2.7.4-alpine
    container_name: Caddy
    network_mode: host
  gitea:
    image: gitea/gitea:1.20.2-rootless
    container_name: Gitea
    ports:
      - "127.0.0.1:4000:3000"
      - "2222:22"
```

Traži se otvoren pristup na portove 443 i 80, a za 2222 se traži dozvola samo za ip 192.168.1.3. *Host* način mreže nije pod utjecajem `DOCKER-USER` lanca pa stoga ulazna pravila specificiramo uobičajeno u `INPUT` lancu, dok pravila za Gitea server specificiramo u `DOCKER-USER` lancu. U odnosu na navedeno, *whitelist* pristup vatrozidu za ovu konfiguraciju izgledao bi ovako:

```
# iptables -P INPUT DENY
# iptables -A INPUT -i lo -j ACCEPT
# iptables -A INPUT -i eth0 -p tcp -m multiport --dports 80,443 -j ACCEPT
# iptables -I DOCKER-USER -i eth0 -p tcp --dport 2222 ! -s 192.168.1.3 -j DROP
```

Dakle, prvotno zabranjujemo sve ulazne (*eng. inbound*) konekcije za `INPUT` lanac. Zatim dozvoljavamo ulazne konekcije, ali samo na internoj mreži računala. Time omogućavamo korištenje Caddy servera kao *reverse proxy* za Gitea server preko porta 4000 otvorenog na internoj mreži računala. Nastavljamo omogućavanjem TCP veza na portove 80 i 443 koje Caddy koristi u `INPUT` lancu zbog *host* mrežnog načina. Naposljetku omogućavamo pristup Gitea serveru na portu 2222 samo s izvorišne adrese 192.168.1.3 (`DOCKER-USER` lanac koristi *blacklist* pristup).

6. Vatrozid

Svakom uređaju spojenom na internet potreban je dobro definiran vatrozid (*eng. firewall*) kako bi se njegovim lokalnim servisima ograničio pristup. Mrežne veze su dvosmjerne, no dijelimo ih na ulazne (*eng. inbound*) i izlane (*eng. outbound*). Ulazne veze su one koje se iniciraju prema našem računalu, dok su izlazne one koje naše računalo inicira prema drugim računalima. Npr. svaki web server čini sadržaj dostupnim na portovima 80 i 443. Promet koji korisnici rade prema tom serveru je njemu ulazni i on ga regulira preko `INPUT` iptables lanca, dok korisnici taj promet reguliraju preko `OUTPUT` iptables lanca.

Linux nudi 2 bazna alata za manipuliranje vatrozida:

iptables
: Iptables [37] je originalna implementacija programa za postavljanje Linux vatrozida. Danas se smatra zastarjelom implementacijom i često je iptables binarna datoteka zapravo simbolična poveznica za *iptables-translate* koji vrši translaciju iptables naredbe u nftables format. Unatoč tome, u ovoj se sekciji razmatra iptables budući da Docker i dalje koristi tu implementaciju vatrozida [36].

nftables
: Nftables [38] je moderna implementacija programa za postavljanje Linux vatrozida te djeluje kao potpuna zamjena za iptables, ip6tables, arptables i ebtables.

Ručno korištenje navedenih alata ponekad može biti komplicirano pa su stoga razvijena razna sučelja (*eng. frontend*) koja internalno koriste te alate. Neka od njih su:

- **ufw** - koriste ga sustavi bazirani na Debian distribuciji
- **firewalld** - koriste ga RedHat sustavi (Fedora, RHEL)

6.1. Regionalno blokiranje

Regionalno blokiranje poželjno je kad se servisi servera žele činit dostupnima samo određenim državama. Alternativno, može se blokirati određene regije ili grupe adresa koje su poznate po čestim napadima. Ako se radi o manjim brojem adresa, pravila za npr. blokiranje mogu se u potpunosti konstruirati korištenjem iptables alata:

```
# iptables -A INPUT --source 192.168.1.3,192.168.1.4 -j DROP
```

Blokiranje većeg broje adresa nepraktično je korištenjem ove metode te se u praksi koristi ipset [39] alat za definiranje setova adresa koje se onda referenciraju u iptables naredbama.

Za ipset je prvo potrebno nabaviti set IP adresa oko kojih želimo stvarati pravila. Najčešće se radi o datoteci s CIDR notacijom IP adresa neke regije. Ovaj primjer koristiti će besplatne ip setove dostupne na Iplocation [40] web stranici. Preuzet će se IP set za Njemačku, odabiremo IPv4 (postoji i IPv6 opcija) i CIDR format.

Stranica vraća `firewall.txt.gz` datoteku koju ju potrebno raspakirati:

```
# gunzip firewall.txt.gz
# head firewall.txt
  # ------------------------------------------------------
  # Free IP2Location Firewall List by Country
  # Source: https://www.ip2location.com/free/visitor-blocker
  # Last Generated: 23 Aug 2023 21:45:50 GMT
  # [Important] Please update this list every month
  # ------------------------------------------------------
  2.16.1.0/24
  2.16.3.0/24
  2.16.6.0/23
  2.16.9.0/24
```

IP2Location uključuje komentare u prvih 6 linija svih CIDR datoteka. Potrebno ih je ukloniti ručno ili npr. koristeći *sed*:

```
# sed -i 1,6d firewall.txt
```

Sljedeći korak je stvoriti ipset u kojeg ćemo učitati CIDR adrese. Učitavanje se vrši koristeći *ipset add* naredbu za svaku adresu pa je praktično koristiti petlju. Ovisno o broju adresa, ova radnja može potrajati.

```
# ipset create Germany hash:net
# for IP in $(cat firewall.txt); do
   ipset add Germany "$IP"
  done
ipset v7.17: Hash is full, cannot add more elements
ipset v7.17: Hash is full, cannot add more elements
ipset v7.17: Hash is full, cannot add more elements
...
```

Nakon nekog vremena ipset javlja kako prije stvoreni *hash net* više nema slobodnih mjesta za preostale adrese.

```
# ipset list | head
Name: Germany
Type: hash:net
Revision: 7
Header: family inet hashsize 32768 maxelem 65536 bucketsize 12 initval 0xeb9e6aa1
Size in memory: 1970736
```

```
References: 0
Number of entries: 65890
Members:
83.231.214.232/29
154.25.7.230
...
# wc -l firewall.txt
92879 firewall.txt
```

Vidimo kako je zadana vrijednost maksimalnog broja adresa 65536, dok ih naš set ima 92879. Potrebno je obrisati set i stvoriti novi uz specifikaciju maksimalnog broja elemenata te ponovno pokrenuti učitavanje:

```
# ipset destroy Germany
# ipset create Germany hash:net maxelem 100000
# for IP in $(cat firewall.txt); do
   ipset add Germany "$IP"
 done
```

Sada je ipset spreman za korištenje pa se vraćamo na iptables. Potrebno je koristiti *set* modul kojeg je moguće specificirati s **-m** opcijom. Iako nije potrebno, ovdje ćemo definirati odvojeni lanac `BLOCKLIST` te mu dodati pripadna pravila. Naposljetku dodajemo `BLOCKLIST` lanac na `INPUT` lanac.

```
# iptables -N BLOCKLIST
# iptables -A BLOCKLIST -m set --match-set Germany src -j DROP
# iptables -A BLOCKLIST -j RETURN
# iptables -A INPUT -j BLOCKLIST
```

Dakle, ulazni paket prvo ide u `INPUT` lanac te dolazi do novostvorenog pravila koje preusmjerava paket u `BLOCKLIST` lanac. Paketi s odredišnom adresom iz Njemačke biti će odbačeni, dok ostali dolaze do drugog pravila lanca koje paket natrag vraća u izvorišni lanac gdje se provodi daljna evaluacija paketa prema definiranim pravilima.

6.2. Odvojen zapisnik

Iptables vatrozid sam po sebi ne održava nikakav zapisnik, već ga je potrebno eksplicitno koristiti. Ovo je moguće korištenjem `LOG` argumenta za **-j** opciju. Kad bi smo npr. blokirali port 22 za ip 192.168.1.13, potrebno je ponoviti pravilo s `LOG` opcijom:

```
# iptables -I INPUT -p tcp --dport 22 -s 192.168.1.13 -j LOG
# iptables -I INPUT -p tcp --dport 22 -s 192.168.1.13 -j DROP
```

Ovo je validan način zapisivanja blokiranih paketa, no može znatno otežati čitanje ostalih elemenata sistemskog zapisnika. Ovo je moguće rješiti koristeći ulogd [41] program za odvojeno logiranje iptables paketa. Zadana konfiguracija dostupna je u `/etc/ulogd.conf`

datoteci. Moguće su razne konfiguracije, a u nastavku su prikazane linije koje je potrebno odkomentirati u konfiguracijskoj datoteci kako bi se dobila konfiguracija koja se koristi u ovoj sekciji:

```
[global]
logfile="/var/log/ulogd.log"
plugin="/usr/lib64/ulogd/ulogd_inppkt_NFLOG.so"
plugin="/usr/lib64/ulogd/ulogd_filter_IFINDEX.so"
plugin="/usr/lib64/ulogd/ulogd_filter_IP2STR.so"
plugin="/usr/lib64/ulogd/ulogd_filter_PRINTPKT.so"
plugin="/usr/lib64/ulogd/ulogd_output_LOGEMU.so"
plugin="/usr/lib64/ulogd/ulogd_raw2packet_BASE.so"
stack=log1:NFLOG,base1:BASE,ifi1:IFINDEX,ip2str1:IP2STR,print1:PRINTPKT,emu1:LOGEMU
```

Prethodni primjer sada bi mogao izgledao ovako:

```
# iptables -I INPUT -p tcp --dport 22 -s 192.168.1.13 -j NFLOG --nflog-prefix '[
    BLOKIRANO-SSH]'
# iptables -I INPUT -p tcp --dport 22 -s 192.168.1.13 -j DROP
```

Ova konfiguracija zapisivati će blokirane pakete u `/var/log/ulogd.log`, a za port 22 će koristiti prefix *[BLOKIRANO-SSH]*. Ovakvo vođenje zapisnika pojednostavljuje pretraživanje te drži sistemski zapisnik čistim. Dodatno, može se koristiti pomoćni alias za uživo pregledavanje novo blokiranih paketa:

```
# alias flog="tail -f /var/log/ulogd.log"
```

Primjer naredbe koja prikazuje broj blokiranih paketa prema portu, silazno sortirano prema broju paketa:

```
# grep "BLOKIRANO" /var/log/ulogd_syslogemu.log  | awk '{print $18}' | cut -d '=' -f
     2 | sort | uniq -c | sort -r | less
```

6.3. Neispravni paketi

Iptables se ne koristi samo za ograničavanje pristupa portovima, već se može i koristiti za filtriranje potencijalno zlonamjernih paketa prije nego li uopće dođu do samih aplikacija. Ovo dovodi do određenog nivoa zaštite od različitih DDoS i ostalih napada. Do sada je bila prikazana manipulacija `INPUT` lanca, no taj lanac nije idealan za filtraciju loših paketa. Prema Tevault [42], neispravne pakete idealnije je blokirati u `PREROUTING` lancu za *mangle* tablicu budući da isti nema *filter* tablicu za razliku od `INPUT` lanca. U nastavku su prikazana pravila za zaštitu od nekih napada.

Neispravni paketi

Blokiramo neispravne pakete koji nisu dio bilo koje postojeće konekcije koristeći *conntrack* modul:

```
# iptables -t mangle -A PREROUTING -m conntrack --ctstate INVALID -j DROP
```

Novi paketi koji nisu SYN

Standardni model TCP veze sa poslužiteljom funkcionira na način da klijent prvo pošalje SYN, dobi SYN+ACK kao odgovor te naposljetku vraća ACK poslužitelju. Ovdje odbacujemo nove konekcije koje ne započinju sa SYN paketom:

```
# iptables -t mangle -A PREROUTING -p tcp ! --syn -m conntrack --ctstate NEW -j
    DROP
```

ICMP paketi

ICMP odgovori nisu nužni za normalan rad servera, a blokiranjem istih preveniraju se *ICMP Flood* napadi.

```
# iptables -t mangle -A PREROUTING -p icmp -j DROP
```

XMAS skeniranje

XMAS skeniranje otvorenih portova bazira se na kombinaciji FIN, PSH i URG tcp zastavica. Lako ga je inicirati koristeći nmap [43] s *-sX* opcijom.

```
# iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP
```

Null paketi

Paketi bez i jedne TCP zastavice. Null skeniranje lako je inicirati koristeći nmap [43] s *-sN* opcijom.

```
# iptables -t mangle -A PREROUTING -p tcp --tcp-flags ALL NONE -j DROP
```

6.4. Ograničavanje konekcija

Ograničavanje broja uspostavljenih konekcija

Ovo pravilo ne dozvoljava više od 100 uspostavljenih konekcija po klijentu. Ovaj broj treba prilagoditi potrebama servera.

```
# iptables -A INPUT -p tcp -m connlimit --connlimit-above 100 -j REJECT --
    reject-with tcp-reset
```

Ograničavanje broja novih konekcija za klijenta po jedinici vremena

Navedena pravila osiguravaju prihvaćanje samo 30 paketa unutar 1 minute. Ako u ovoj jedinici vremena dođe više nego 30 paketa, oni se odbacuju. Nakon 1 minute brojač se resetira i opet je dozvoljeno 30 paketa u toj minuti.

```
# iptables -A INPUT -p tcp -m conntrack --ctstate NEW -m limit --limit 60/s --
    limit-burst 30 -j ACCEPT
# iptables -A INPUT -p tcp -m conntrack --ctstate NEW -j DROP
```

6.5. Skriptiranje

Menadžment iptables pravila nije jednostavan te može dovesti do zaključavanja iz vlastitog servera. Ovdje stvaramo pomoćnu skriptu s vlastitim lancima koja vrši postupno postavljanje vatrozida, može se pokrenuti neograničen broj puta jer uvijek ostavlja isto stanje (*eng. idempotent*) te ne prekida trenutnu SSH vezu. Redoslijed aktivnosti je sljedeći:

1. Determiniraj lokalno sučelje i adresu
2. Postavi zadanu vrijednost za `INPUT` lanac na `ACCEPT` kako ne bi došlo do prekida veza kod kasnijeg čišćenja
3. Obriši sadržaj svih lanaca
4. Obriši sve ipset regije
5. Učitaj sve ipset regije. Direktorij *ipset* nalazi se u istom direktoriju gdje i ova skripta, a sadrži tekstualne datoteke s CIDR adresama. Ime datoteke se koristi za kasnije referenciranje u pravilima.
6. Stvori nove lance. Ovo je korisno za enkapsulaciju logičnih cjelina i jednostavnije održavanje. Opis lanaca:
   - `LOKALNO` - odnosi se na promet lokalne mreže. Ovdje eksplicitno prihvaćamo pakete samo za port 22 dok ostale odbacujemo.
   - `REGIONALNO` - lanac koji propušta samo pakete iz Njemačke dok ostale odbacuje
   - `NEISPRAVNO` - odbacuje razne neispravne pakete
   - `SHARED` - budući da ulazni promet za aplikacije van Docker kontejnera (ili one u kontejneru koje koriste *host* mrežni način) dolazi u `INPUT` lanac, a za Docker kontejnere unutar `DOCKER-USER` lanca, stvaramo `SHARED` lanac u kojeg dodajemo pravila koja vrijede za oba lanca. `SHARED` lanac uključuje `REGIONALNO` i `NEISPRAVNO` lanac.
   - `SERVER-INPUT` - pomoćni lanac kojeg kasnije dodajemo na `INPUT` lanac. Uključuje `LOKALNO` i `SHARED` lanac, dozvoljava port 22 te portove 80 i 443 za webserver kontejner koji koristi *net* način mreže.
   - `SERVER-DOCKER` - pomoćni lanac kojeg kasnije dodajemo na `DOCKER-USER` lanac. Uključuje `SHARED` lanac te dozvoljava port 2222 za Gitea aplikaciju u kontejneru.
7. Novostvorene lance spoji na postojeće zadane lance.
8. Spremi ipset i iptables konfiguraciju koja se koristi za ponovno učitavanje kod pokretanja sustava. Omogućivanje servisa za učitavanje ovih datoteka ovisi o distribuciji.

Skripta:

```
#!/bin/sh

# Determiniranje sucelja i adrese
#############################################################################
NIC="$(ip link | head -n 3 | tail -n 1 | cut -d ':' -f 2)"
Local_IP="$(ip -4 -o addr | grep "$NIC" | awk '{print $4}')"

# Čšćienje
#############################################################################
iptables -P INPUT   ACCEPT
iptables -P OUTPUT  ACCEPT
iptables -P FORWARD DROP

iptables -F INPUT
iptables -F OUTPUT
iptables -t mangle -F

iptables -F DOCKER-USER

iptables -F SERVER-INPUT
iptables -X SERVER-INPUT
iptables -F SERVER-DOCKER
iptables -X SERVER-DOCKER
iptables -F SHARED
iptables -X SHARED
iptables -F NEISPRAVNO
iptables -X NEISPRAVNO
iptables -F REGIONALNO
iptables -X REGIONALNO
iptables -F LOKALNO
iptables -X LOKALNO

# Postavljanje ipset
#############################################################################
ipset destroy

for COUNTRY in $(ls ipsets); do
  maxelem="$(wc -l "$COUNTRY")"
  ipset -q create "$COUNTRY" hash:net maxelem "$maxelem"
  for IP in $(cat "ipsets/$COUNTRY"); do
    ipset -q add "$COUNTRY" "$IP"
  done
done

# Novi lanci
#############################################################################

# LOKALNO
iptables -N LOKALNO
iptables -A LOKALNO -i lo   -j ACCEPT
iptables -A LOKALNO -i $NIC -p tcp -s $Local_IP --dport 22 -j ACCEPT
iptables -A LOKALNO -i $NIC -s $Local_IP -j NFLOG --nflog-prefix "[DROP-LOKALNO]"
iptables -A LOKALNO -i $NIC -s $Local_IP -j DROP
```

```
iptables -A LOKALNO -j RETURN

# REGIONALNO
iptables -N REGIONALNO
iptables -A REGIONALNO -m set   --match-set Germany src -j RETURN
iptables -A REGIONALNO -j NFLOG --nflog-prefix '[DROP-INTRUDER]'
iptables -A REGIONALNO -j DROP

# NEISPRAVNO
iptables -N NEISPRAVNO
iptables -A NEISPRAVNO -m conntrack --ctstate INVALID -j DROP
iptables -A NEISPRAVNO -p tcp ! --syn -m conntrack --ctstate NEW -j DROP
iptables -A NEISPRAVNO -p icmp -j DROP
iptables -A NEISPRAVNO -p tcp --tcp-flags ALL FIN,PSH,URG -j DROP
iptables -A NEISPRAVNO -p tcp --tcp-flags ALL NONE -j DROP
iptables -A NEISPRAVNO -p tcp --tcp-flags ALL NONE -j DROP
iptables -A NEISPRAVNO -j NFLOG --nflog-prefix '[DROP-NEISPRAVNO]'
iptables -A NEISPRAVNO -j RETURN

# SHARED
iptables -N SHARED
iptables -A SHARED -i $NIC -m state --state NEW -j REGIONALNO
iptables -A SHARED -i $NIC -j NEISPRAVNO
iptables -A SHARED -j RETURN


# SERVER-INPUT
iptables -N SERVER-INPUT
# Propusti vec uspostavljene konekcije
iptables -A SERVER-INPUT -i $NIC -m conntrack --ctstate ESTABLISHED,RELATED -j
    ACCEPT
# Spajanje
iptables -A SERVER-INPUT -j LOKALNO
iptables -A SERVER-INPUT -j SHARED
# SSH
iptables -A SERVER-INPUT -i $NIC -p tcp --dport 22 -j ACCEPT
# Caddy
iptables -A SERVER-INPUT -i $NIC -p tcp --dport 80 -j ACCEPT
iptables -A SERVER-INPUT -i $NIC -p tcp --dport 443 -j ACCEPT
#
iptables -A SERVER-INPUT -i $NIC -j NFLOG --nflog-prefix '[DROP-INPUT]'
iptables -A SERVER-INPUT -i $NIC -j DROP


# SERVER-DOCKER
iptables -N SERVER-DOCKER
# Propusti vec uspostavljene konekcije
iptables -A SERVER-DOCKER -i $NIC -m conntrack --ctstate ESTABLISHED,RELATED -j
    ACCEPT
# Spajanje
iptables -A SERVER-DOCKER -j SHARED
# Gitea
iptables -A SERVER-DOCKER -i $NIC -p tcp --dport 2222 -j RETURN
```

```
#
iptables -A SERVER-DOCKER -i $NIC -j NFLOG --nflog-prefix '[DROP-DOCKER]'
iptables -A SERVER-DOCKER -i $NIC -j DROP


# Primjena
################################################################################

# PREROUTING
iptables -t mangle -A PREROUTING -j NEISPRAVNO

# INPUT
iptables -A INPUT -j SERVER-INPUT

# DOCKER-USER
iptables -A DOCKER-USER -j SERVER-DOCKER
iptables -A DOCKER-USER -j RETURN


# Spremanje
################################################################################
ipset    save > /etc/iptables/ipsets.txt
iptables-save > /etc/iptables/iptables.txt
sync
```

7. Jezgra

Linux na prvi pogled djeluje kao veoma siguran sustav koji je besplatan, otvorenog koda i brz, no realnost je puno kompliciranija. Naime, Linux jezgra koristi monolitnu arhitekturu zbog koje je cijela jezgra svojevrsno jedan veliki process s root privilegijama, a na Linux sustavima root nema granica. Linux jezgra je masivna i ima preko 8 milijuna [44] linija koda, a greška u bilo kojem dijelu koda potencijalni je vektor napada. Zbog ovih problema, DanielMicay [45], autor GrapheneOS Android sustava, navađa kako mu je jedan od dugoročnih ciljeva zamijeniti Linux jezgru GrapheneOS sustava s alternativnom koja bi koristila *microkernel* arhitekturu. Vyukov [46] navađa dodatne probleme: često dodavanje novih značajka bez adekvatnog testiranja, spor proces uključivanja sigurnosnih zakrpa u LTS verzije kernela, neadresirani prijavljeni problemi itd. Većina korisnika nije uključena u razvoj jezgre, no i dalje može značajno minimizirati šanse uspješnih napada minimiziranjem aktivnih dijelova Linux jezgre koje ne koriste ili podešavanjem različitih postavka. Linux jezgra detaljno se može konfigurirati prilikom kompilacije, no to je veoma nepraktičan i često dugotrajan proces. Ovo poglavlja razmatra 3 načina manipulacije Linux jezgre bez kompilacije:

- **Sysctl** - konfiguriranje /proc virtualnog datotečnog sustava za izmjenu postavka
- **Cmdline** - dodatno učvršćivanje dodavanjem argumenata jezgri prilikom pokretanja sustava
- **Moduli** - onemogućavanje nepotrebnih jezgrenih modula

7.1. Sysctl

Linux jezgra održava virtualni sustav u `/proc` direktoriju. Kad bi željeli npr. isključiti vremenske oznake TCP paketa (*tcp_timestamps*), to možemo učiniti:

- **pisanjem u datoteku**

```
# echo 0 > /proc/sys/net/ipv4/tcp_timestamps
```

- **koristeći sysctl**

```
# sysctl net.ipv4.tcp_timestamps=0
```

Ručno pisanje u datoteku treba se izbjegavati budući da je nepregledno, već se treba preferirati korištenje sysctl alata. Postavke koje se ovako mijenjaju vrijede samo za vrijeme rada sustava te se resetiraju na zadane vrijednosti prilikom ponovnog pokretanja. Kako bi izmjene učinili perzistentnima, moguće ih je pohraniti u `/etc/sysctl.conf` datoteku. Neke distribucije nude odvojen `/etc/sysctl.d` direktorij u koji se mogu pohraniti konfiguracijske datoteke s *.conf* sufiksom, a sam sadržaj sysctl konfiguracijskih datoteka je zapravo niz *ključ-vrijednost* (*eng. key-value*) parova. Ova sekcija razmatrat će samo nekoliko tih opcija budući da ih ima veoma puno te ovise o potrebama korisnika.

Onemogućavanje pokretanje druge jezgre

Kexec omogućava pokretanje druge jezgre za vrijeme rada sustava. Ovo se ponekad koristi za pokretanje druge jezgre koja može uhvatiti stanje primarne kad ona naiđe na neispravljivu grešku (omogućava *debugging*) [47]. Problem je da ovo omogućava učitavanje zlonamjerne jezgre pa je bolje isključiti kexec ako nije potreban.

```
# sysctl kernel.kexec_load_disabled=1
```

Skrivanje pokazivača jezgre

Napadač može dobiti korisne informacije o jezgri koje mogu pomoć kod eksploatacije [48].

```
# sysctl kernel.kptr_restrict=2
```

Onemogućavanje neprivilegiranog eBPF podsustava

eBPF podsustav sadrži velik broj sigurnosnih problema zbog kojih je neprivilegiran eBPF na nekim distribucijama onemogućen prema zadanim vrijednostima [49]. Ova opcija obično se kombinira s drugom za učvršćivanje bpf JIT kompajlera.

```
# sysctl kernel.unprivileged_bpf_disabled=1
# sysctl net.core.bpf_jit_harden=2
```

Onemogućavanje TCP SACK

Prema Netflix [50], TCP SACK može se koristiti za stvaranje jezgrene panike (*eng. kernel panic*) na udaljenom računalu pa ga je najbolje onemogućiti.

```
# sysctl net.ipv4.tcp_sack=0
# sysctl net.ipv4.tcp_fack=0
# sysctl net.ipv4.tcp_dsack=0
```

Omogućavanje syncookie značajke

Syncookie zaštićuje sustav od *TCP SYN flood* DDoS napada [51].

```
# sysctl net.ipv4.tcp_syncookies=1
```

Zaštita od lažiranja izvorišne adrese

Ova opcija validira izvorišne adrese zahtjeva i time zaštićuje od prihvaćanja paketa koji npr. lažira da se nalazi na lokalnoj mreži.

```
# sysctl net.ipv4.conf.all.rp_filter=1
# sysctl net.ipv4.conf.default.rp_filter=1
```

Onemogućavanje ptrace funkcionalnosti

Ptrace omogućava programima detaljan pregled rada drugih programa, pa tako i njihove memorije što omogućava velik spektar potencijalnih napada [52].

```
# sysctl kernel.yama.ptrace_scope=2
```

Onemogućavanje korištenja procesorske random funkcije

Moderni procesori imaju vlastiti generator nasumičnih brojeva, no on je potencijalno nesiguran te se generalno ne može verificirati zbog zatvorene prirode [53].

```
# sysctl random.trust_cpu=off
```

7.2. Cmdline parametri

Prilikom pokretanja Linux jezgre, mogu joj se dodati argumenti kojima se mijenjaju različiti dijelovi sustava. Cmdline argumenti trenutno pokrenutog sustava vidljivi su u `/proc/cmdline` datoteci (primjer za Alpine Linux):

```
# cat /proc/cmdline
BOOT_IMAGE=/boot/vmlinuz-lts modules=loop,squashfs,sd-mod,usb-storage quiet  initrd
    =/boot/initramfs-lts
```

Prethodna naredba prikazuje samo argumente koji su bili eksplicitno dodani. Zadane vrijednosti argumenata koje su bile postavljene prije kompilacije Linux jezgre ovdje neće biti vidljive, već ih je potrebno potražiti u jednoj od navedenih datoteka (ovisi o distribuciji):

- `/proc/config.gz` - potrebno prvo dekompresirati s npr. *zcat* naredbom
- `/boot/config*` - moguće izravno pregledavati s npr. *less* naredbom

Dodavanje argumenata ovisi o korištenom *bootloader-u*. Najčešće korišteni je GRUB, a konfigurira se putem `/etc/default/grub` datoteke. Relevantna varijabla datoteke je `GRUB_CMDLINE_LINUX_DEFAULT` te argumente dodajemo u nju. Kao i za *sysctl*, postoji velik broj mogućih argumenata [54] pa ih je ovdje navedeno samo nekoliko.

Osiguravanje memorije

Ovaj argument smanjuje mogućnost eksploatacije *Use-after-free* [55] klase ranjivosti koja se bazira na konceptu pristupa prije otpuštenoj memoriji koja još uvijek sadržava senzitivan sadržaj. Prvi argument radi na principu ispisivanja alocirane memorije s nulama prije nego ona bude otpuštena, dok drugi ispisuje memoriju s nulama prije nego ona bude alocirana.

```
init_on_free=1 init_on_alloc=1
```

Dozvoli samo module s validnim potpisom

Otežava učitavanje zlonamjernih modula za jezgru [56].

```
module.sig_enforce=1
```

Onemogućavanje debugfs prostora

Debugfs je virtualni datotečni sustav koristan programerima Linux jezgre za otkrivanje grešaka, no može sadržavati senzitivne informacije zbog kojeg ga je najbolje onemogućiti ako nije potreban [57].

```
debugfs=off
```

Ograničavanje pristupa jezgri

Root je generalno svemoguć, no nema potpun pristup svim aspektima jezgre. Ove vrijednosti argumenta dodatno ograničavaju root [58]:

- integrity - isključivanje komponenata koje omogućavaju modificiranje jezgre kao što je *kexec*
- confidentiality - radi sve što i integrity s dodatkom ograničavanja pristupa senzitivnim informacijama jezgre

```
lockdown=confidentiality
```

Onemogućavanje IPv6

Ako Ipv6 nije potreban, može se isključiti.

```
ipv6.disable=1
```

7.3. Moduli

Jezgreni moduli dodavaju nove značajke jezgri te se dinamički mogu uključivati i isključivati iz jezgre. Moduli se nalaze u `/usr/lib/modules/${uname-r}` direktoriju. Trenutno aktivne module lako je moguće vidjeti:

```
# lsmod
Module                  Size  Used by
xt_conntrack           16384  6
nf_conntrack          184320  3 xt_conntrack,nf_nat,xt_MASQUERADE
nf_defrag_ipv6         24576  1 nf_conntrack
nf_defrag_ipv4         16384  1 nf_conntrack
xt_CHECKSUM            16384  112
xt_tcpudp              20480  0
...
```

Moduli se po potrebi dinamički učitavaju, no u nekim slučajevima potrebno ih je ručno omogućiti. Omogućavanje modula do sljedećeg pokretanja računala moguće je sa *modprobe*, a isključivanje s *modprobe -r* (osim ako je modul trenutno korišten):

```
# modbrope tcp_bbr
# modprobe -r tcp_bbr
modprobe: FATAL: Module tcp_bbr is in use.
```

Svaki uključen modul dovodi do više aktivnog koda što dovodi do potencijalno veće površine napada. Iz ovog je razloga preporučljivo isključiti module koji nisu potrebni, a isto se vrši dodavanjem modula u blacklist datoteke sa *.conf* sufiksom koje se nalaze u `/etc/modprobe.d/` direktoriju. Npr. ako želimo onemogućiti *bluetooth*, potrebno je isključiti *bluetooth* i *btusb* module. Stvaramo datoteku `/etc/modprobe.d/blacklist.conf` sa navedenim sadržajem:

```
blacklist bluetooth
blacklist btusb
```

Ovaj pristup onemogućuje automatsko uključivanje tih modula, no ne sprječava ručno učitavanje s *modprobe*. Za potpuno onemogućavanje modula, moguće je definirati prilagođenu naredbu koja će se izvršiti umjesto zadane (učitavanje modula) s *install* deklaracijom nakon

koje slijedi ime modula i zamjenski program. Ovdje će se koristiti `/bin/false` koji vraća status greške i ne radi ništa drugo:

```
install bluetooth /bin/false
install btusb /bin/false
```

Ne postoji definitivna lista modula koji bi se trebali onemogućiti budući da to u potpunosti ovisi o korištenim značajkama, no u nastavku su ponuđeni neki moduli koji su vrijedni razmatranja za smanjenje potencijalnih vektora napada.

USB uređaji

Server će najčešće koristiti SSD diskove kao pohrambene medije, dok USB uređaji generalno nisu potrebni. Kako bi se prevenirali različiti USB napadi, moguće je isključiti module za USB, firewire i thunderbolt priključke:

```
install usb-storage   /bin/false
install thunderbolt   /bin/false
install firewire-core /bin/false
```

Nekorišteni mrežni protokoli

Dostupan je velik broj različitih mrežnih protokola koji se u praksi gotovo nikad ne koriste, a sadržavaju potencijalne ranjivosti.

```
install ax25 /bin/false
install netrom /bin/false
install x25 /bin/false
install rose /bin/false
install sctp /bin/false
install rds /bin/false
install tipc /bin/false
install n-hdlc /bin/false
install decnet /bin/false
install p8022 /bin/false
install can /bin/false
install econet /bin/false
install af_802154 /bin/false
install ipx /bin/false
install dccp /bin/false
install psnap /bin/false
install p8023 /bin/false
```

Bluetooth

Bluetooth nije potreban na serverima, a njegova povijest puna je različitih ranjivosti [59].

```
install bluetooth /bin/false
install btusb     /bin/false
```

8. Web server

Web server esencijalna je komponenta svakog servera i izbor je široki. Nginx i Apache najpopularniji su web serveri koji nude visoku razinu prilagodljivosti, no sve više je popularan Caddy. Naime, Nginx i Apache zahtjevaju podosta konfiguracije za najjednostavnije stvari, a dodatno ih je potrebno podešavati kako bi koristili najbolje sigurnosne prakse (npr. korištenje samo TLS 1.2 i 1.3 verzije, automatska redirekcija na sigurnu HTTPS vezu). Caddy ima ugrađene najbolje sigurnosne prakse te dodatno vrši i automatsko postavljanje i obnovu LetsEncrypt certifikata koristeći ACME protokol, a i konfiguracija je veoma jednostavna. Dodatno, Caddy se razlikuje od drugih servera po korištenju Go jezika zbog kojeg su smanjene mogućnosti neke potencijalne ranjivosti u smislu grešaka s memorijom.

U navedenim sekcijama demonstriraju se neke mogućnosti Caddy servera preko Caddyfile datoteke koja se nalazi na `/etc/caddy/Caddyfile` putanji.

8.1. Certifikati

Caddy automatski pribavlja certifikate za domene, no moguće je specificirati alternativne lokalne certifikate s *tls* direktivom.

Npr. pribavljanje LetsEncrypt certifikata jednostavno je koristeći *Certbot*. Navedena naredba pribavlja *wildcard* certifikat koji vrijedi za domenu i sve njene poddomene. Za ovakav certifikat potrebno je izvršiti HTTP server i DNS izazov za dokazivanje vlasništva domene i poddomena.

```
# certbot certonly --agree-tos --manual           \
                   -d domena.com -d *.domena.com  \
                   --rsa-key-size 4096
...
```

Pribavljeni certifikati spremaju se u `/etc/letsencrypt/live/domena.com` direktorij. U Caddyfile specificiramo *tls* direktivu za domenu:

```
domena.com {
  tls /etc/letsencrypt/live/domena.com/fullchain.pem /etc/letsencrypt/live/domena.
      com/privkey.pem
}
```

8.2. Reverse proxy

Server obično nudi više različitih servisa. Kako bi se svakom servisu pristupalo preko sigurnosne veze, nije praktično svaki od njih konfigurirati individualno, a i nekad je nemoguće ako ta aplikacije to ne nudi. Rješenje ovog problema je korištenje web servera kao *reverse proxy* gdje on procesira sve veze i certifikate nužne za HTTPS, a dekriptiran promet povezuje s lokalnim aplikacijama.

S Caddy web serverom, reverse proxy je veoma jednostavan. Kad bi željeli sav promet na `domena.com` preusmjeriti na lokalnu aplikaciju koja je dostupna na portu 1234:

```
domena.com {
        reverse_proxy http://127.0.0.1:1234
}
```

U slučaju da se jedna aplikacija nalazi na podputanji `/app1`, a druga na `/app2`, a obje želimo servirati na istoj poddomeni:

```
aplikacije.domena.com {
        reverse_proxy /app1/* http://127.0.0.1:1234
        reverse_proxy /app2/* http://127.0.0.1:4321
}
```

8.3. File server

Caddy može činiti sadržaj direktorija definiranog s *root* direktivom javno dostupnim sa *file_server* deklaracijom. Prvo se definira direktorij, a zatim *file_server* deklaracija. Ako se želi nuditi mogućnost vizualnog pregleda direktorija i svih pod direktorija, dodaje se *browse* argument *file_server* deklaraciji.

```
domena.com {
       root * /srv/web_stranica
       file_server
}
```

8.4. Rad s putanjama

Ponekad je potrebna kompleksnija konfiguracija putanja. Uzmimo primjer u kojem:

- poslužujemo web stranicu na `domena.com`
- njenu aplikaciju na `domena.com/app1` bez da `/app1` dio putanje šaljemo samoj aplikaciji
- njenu drugu aplikaciju na `/app2` s tim da dodajemo `/action/1` putanji prije slanja aplikaciji
- na `/files` nudimo pristup nekim datotekama s mogučnošću pregledavanja

Pripadni Caddyfile:

```
domena.com {
  handle_path /app1/* {
    uri strip_prefix /app1
    reverse_proxy http://127.0.0.1:1234
  }
  handle_path /app2/* {
    rewrite * /action/1
    reverse_proxy http://127.0.0.1:4321
  }
  handle_path /files {
    file_server browse
  }
  file_server
}
```

Za putanje se koristi *handle_path* direktiva. Direktiva *file_server* nalazi se tek na kraju budući da djeluje kao zadana radnja u slučaju da zahtjev ne odgovara prethodnim *handle_path* putanjama.

8.5. Basic auth

Basic auth omogućava jednostavnu autentifikaciju serverskih aplikacija, no nije idealno rješenje budući da svaki HTTP zahtjev mora sadržavati korisničko ime i lozinku bez ikakvog sažimanja.

Caddy ne podržava izravno zapisivanje lozinka, već ih je prvo potrebno obraditi funkcijom sažimanja:

```
# caddy hash-password
Enter password: ***
Confirm password: ***
$2a$14$MG8GReapbLfJvCgGqEX6FeM108lrEHXR9VQ8Jx3qhWxHVhdzqfqGm
```

Ako želimo osigurati `/admins` putanju i učiniti ju dostupnom samo za korisnika *korisnik123* s prethodnom lozinkom:

```
domena.com {
  basicauth /admins/* {
    korisnik123 $2a$14$MG8GReapbLfJvCgGqEX6FeM108lrEHXR9VQ8Jx3qhWxHVhdzqfqGm
  }
}
```

9. Ostalo

9.1. Skrivanje procesa

Prema zadanim vrijednostima, svaki korisnik može vidjeti tuđe procese, uključujući i procese root korisnika. Ne vide se samo procesi, već i argumenti pozvanih programa:

```
$ ps aux
2429 root      0:00 /sbin/getty 38400 tty6
2432 root      0:00 /sbin/getty -L 0 ttyS0 vt100
2434 root      0:00 -ash
2702 root      0:00 /sbin/syslogd -t -n
3291 korisnik  0:00 ash
3294 korisnik  0:00 ps aux
...
```

Ove informacije mogu biti korisne napadačima u planiranju napada pa ih je bolje sakriti, a to je moguće ponovnim montiranjem `/proc` direktorija s *hidepid* opcijom:

```
# mount -o remount,hidepid=2,gid=polkitd
```

Kako bi ove promjene bile perzistentne, potrebno je izmijeniti `/etc/fstab` datoteku dodavanjem navedene linije [42]:

```
proc    /proc    proc    hidepid=2   0    0
```

9.2. Sistemsko vrijeme

Sistemsko vrijeme od velike je sigurnosne važnosti budući je to temelj za enkriptirane konekcije za koje su potrebni certifikati koji se temelje na vremenu (imaju definirani vremenski raspon u kojem vrijede). Netočno sistemsko vrijeme može rezultirati u velikom broju problema.

Razmotrimo primjer u kojem server nema statičnu ip adresu, već koristi DDNS (*eng. Dynamic DNS*). Ažuriranje ip adrese moguće je koristeći *ddclient* [60] program. Razmotrimo sljedeću konfiguraciju dostupnu u `/etc/ddclient/ddclient.conf` datoteci:

```
daemon=30
ssl=yes

protocol=dyndns2
server=api.dynu.com
login=email
password=***
```

Navedena konfiguracija koristi *ssl=yes* opciju koja osigurava korištenje sigurne veze kod provođenja zahtjeva za ažuriranje adrese. U slučaju da je sistemsko vrijeme servera s

ovom ddclient konfiguracijom krivo, ddclient ne bi mogao ažurirati ip adresu što bi dovelo do nedostupnosti servera. Ako se radi o nekom VPS serveru, i dalje je moguće spojiti se preko njihove web konzole, no ako se radi o vlastitom sklopovlju, ispravak bi zahtjevao putovanje do lokacije servera i ručno spajanje.

Alternativno, krivo sistemsko vrijeme može dovesti do eksploatacije. Malhotra, Cohen, Brakke i dr. [61] demonstrirali su nekoliko NTP napada preko kojih napadač može:

- izmijeniti sistemsko vrijeme NTP klijenta
- onemogućiti ažuriranje sistemskog vremena NTP klijenta

Jedan način osiguravanja NTP servera korištenje je NTS [62] (*eng. Network Time Security*) protokola. Uz to, poželjno je koristiti više NTP servera paralelno u slučaju da je jedan od njih kompromitiran. Ovakva konfiguracija moguća je koristeći Chrony [63] NTP klijent. Primjer ovakve konfiguracije nudi GrapheneOS tim koji ga koristi na vlastitim serverima [64]:

```
server time.cloudflare.com iburst nts
server ntppool1.time.nl iburst nts
server nts.netnod.se iburst nts
server ptbtime1.ptb.de iburst nts

minsources 2
authselectmode require

driftfile /var/lib/chrony/drift
ntsdumpdir /var/lib/chrony

leapsectz right/UTC
makestep 1.0 3

rtconutc
rtcsync

cmdport 0
```

Navedena konfiguracija koristi samo NTP servere koji podržavaju NTS. Od 4 definirana servera, *minsources 2* opcija osigurava ažuriranje samo u slučaju dostupnosti bar 2 NTP servera.

Chrony ponekad nije dovoljan kod korištenja računala bez RTC (*eng. Real Time Clock*) sata. Ovdje se uglavnom radi o SBC (*eng. Single Board Computer*) uređajima. Naime, ako se ddclient koristi sa *ssl=yes* opcijom i na početku rada dobije krivo sistemsko vrijeme (prije nego ga chrony ažurira), moguće je da ne pokuša ponovno ažuriranje. Približno točno vrijeme prilikom pokretanja sustava na takvim sustavima moguće je koristeći *fake-hwclock* [65] program. Potrebno je omogućiti *Crontab* ili sličan mehanizam za periodično pozivanje *fake-hwclock save* naredbe koja sprema trenutno vrijeme u `/etc/fake-hwclock.data` datoteku. Nakon navedene konfiguracije i omogućavanja fake-hwclock servisa, vrijeme će biti približno točno kod svakog ponovnog pokretanja.

9.3. SSH

SSH je standardni način uspostavljanja sigurne veze sa serverom koji dolazi s dobrim zadanim postavkama, no moguće ga je učvrstiti. Konfiguracija se vrši preko `/etc/ssh/sshd_config` datoteke.

Ograničavanje korisnika

Moguće je specificirati korisnike za koje je SSH dozvoljen.

```
AllowUsers korisnik1 korisnik2
```

Onemogućavanje root prijave

Bolje je imati običan korisnički račun te prema potrebi koristiti *sudo* za eskalaciju privilegija.

```
PermitRootLogin no
```

Promjena porta

SSH koristi port 22 prema zadanim vrijednostima. Iako promjena porta ne povećava sigurnost, smanjuje broj zapisa neuspjelih pokušaja u sistemskom zapisniku. Naime, postoji puno automatiziranih malicioznih skripta koje kontinuirano pokušavaju uspostaviti vezu na portu 22.

```
Port 9999
```

Onemogućavanje autentifikacije sa lozinkom

Lozinke koje korisnici odabiru često nisu dovoljno sigurne pa ih je najbolje onemogućiti. Autentifikacija se onda provodi na temelju SSH ključeva. Svaki korisnik servera može specificirati dozvoljene javne ključeve u `~/.ssh/authorized_keys` datoteci.

```
PasswordAuthentication no
KbdInteractiveAuthentication no
```

Automatska odjava nakon nekog vremena

Nakon uspostave SSH veze, neki će korisnici ostaviti vezu otvorenu s računalom bez nadzora. Ova postavka osigurava automatsko zatvaranje veze nakon 10 minuta neaktivnosti.

```
ClientAliveInterval 600
ClientAliveCountMax 1
```

9.4. Hardened malloc

Malloc je zapravo funkcija koja vrši alokaciju memorije te vraća pokazivač na istu. Linux je napisan u C jeziku koji nudi više različitih implementacija standardne biblioteke (glibc, musl itd.), a svaka od njih ima vlastitu implementaciju *malloc* funkcije. Rad s memorijom senzitivna je operacija koja lako može dovesti do različitih napada [66] u slučaju loše implementacije. *Hardened malloc* [67] jedna je takva implementacija *malloc* funkcije koja se može koristiti na Android i Linux sustavima, a nudi dodatnu razinu zaštite protiv raznih memorijskih napada.

Hardened malloc može se koristi samo za specifične aplikacije ili za cijeli sustav. Ako se želi koristiti npr. *cat* naredba s hardened malloc:

```
# LD_PRELOAD="/usr/lib/libhardened_malloc.so" cat datoteka.txt
```

Kako bi koristili hardened malloc za cijeli sustav, potrebno je definirati putanju te biblioteke u `/etc/ld.so.preload` datoteci:

```
/usr/local/lib/libhardened_malloc.so
```

GrapheneOS tim [67] navađa kako je preporučljivo postaviti navedenu *sysctl* opciju zbog načina rada biblioteke:

```
vm.max_map_count = 1048576
```

10. Zaključak

Postavljanje i osiguravanje Linux servera nije lak posao, posebice ako se koristi vlastito sklopovlje. Zbog navedenih su razloga sve popularniji PaaS (*eng. Platform as a Service*) servisi koji obavljaju svo postavljanje i osiguravanje Linux sustava, dok korisnici samo trebaju prebaciti svoje aplikacije na te servise te ih pokrenuti.

Ipak, model vlastitog održavanja Linux servera privlačna je opcija za one koji žele prilagođenu konfiguraciju i veću razinu sigurnosti, no zahtjeva više znanja i iskustva u polju sigurnosti. Ni jedan server nije 100% siguran, no uz adekvatnu konfiguraciju može se ostvariti visoka razina sigurnosti. Dodatno, važno je razumijeti kako se sfera sigurnosti Linux sustava kontinuirano mijenja zbog čega je uvijek potrebno biti u koraku s modernim praksama kako bi održali željenu razinu sigurnosti. Uz sve navedeno, sigurnosni propusti i dalje se mogu manifestirati u korisničkim aplikacijama zbog čega je važno iste pokretati s minimalnim privilegija te unutar nekog kontejnera kako bi se potencijalna šteta minimizirala.

Popis literature

[1] R. tim, *High Performance Cloud VPS Hosting*, 2023. adresa: https://ramnode.com (pogledano 28. 8. 2023.).

[2] arcticblue, *Ramnode hacked. Names, emails, and passwords compromised.* 2013. adresa: https://www.reddit.com/r/webdev/comments/1gga3n/ramnode_hacked_names_emails_and_passwords (pogledano 28. 8. 2023.).

[3] Ramnode, *Ramnode Twitter post*, 2013. adresa: https://twitter.com/RamNode/status/346256439674810369 (pogledano 28. 8. 2023.).

[4] MITRE, *CWE-256: Plaintext Storage of a Password*, 2023. adresa: https://cwe.mitre.org/data/definitions/256.html (pogledano 28. 8. 2023.).

[5] BleepingComputer, *GoDaddy: Hackers stole source code, installed malware in multi-year breach*, 2023. adresa: https://www.bleepingcomputer.com/news/security/godaddy-hackers-stole-source-code-installed-malware-in-multi-year-breach/ (pogledano 28. 8. 2023.).

[6] Groovypost, *https://www.groovypost.com/news/dreamhost-wordpress-hacked*, 2012. adresa: https://www.groovypost.com/news/dreamhost-wordpress-hacked/ (pogledano 28. 8. 2023.).

[7] PCMag, *Hostinger Security Breach Impacts 14M Customers*, 2019. adresa: https://www.pcmag.com/news/hostinger-security-breach-impacts-14m-customers (pogledano 28. 8. 2023.).

[8] M. E. Peck, *Thousands of Bitcoins stolen in a hack on Linode*, 2012. adresa: https://spectrum.ieee.org/thousands-of-bitcoins-stolen-in-a-hack-on-linode (pogledano 28. 8. 2023.).

[9] Slashdot, *Linode hacked, CCs and passwords leaked*, 2013. adresa: https://slashdot.org/firehose.pl?op=view&type=submission&id=2603667 (pogledano 28. 8. 2023.).

[10] Bloomberg, *Shopify Says 'Rogue' Employees Stole Data From Merchants*, 2020. adresa: https://www.bloomberg.com/news/articles/2020-09-22/shopify-says-rogue-employees-stole-data-from-merchants (pogledano 28. 8. 2023.).

[11] S. SN, *Breaking Full Disk Encryption from a Memory Dump*, 2018. adresa: https://blog.appsecco.com/breaking-full-disk-encryption-from-a-memory-dump-5a868c4fc81e (pogledano 28. 8. 2023.).

[12] C. Cimpanu, *20 VPS providers to shut down on Monday, giving customers two days to save their data*, 2019. adresa: https://www.zdnet.com/article/20-vps-providers-to-shut-down-on-monday-giving-customers-two-days-to-save-their-data (pogledano 28. 8. 2023.).

[13] L. H. Newman, *Intel Chip Flaws Leave Millions of Devices Exposed*, 2023. adresa: https://www.wired.com/story/intel-management-engine-vulnerabilities-pcs-servers-iot (pogledano 28. 8. 2023.).

[14] corna, *me$_{cleaner}$*, 2018. adresa: https://github.com/corna/me_cleaner (pogledano 28. 8. 2023.).

[15] A. Hodzic, *DebConf 14: QA with Linus Torvalds*, 2014. adresa: https://youtu.be/5PmHRSeA2c8?t=4075 (pogledano 28. 8. 2023.).

[16] A. L. Tim, *Alpine Linux*, 2023. adresa: https://alpinelinux.org/ (pogledano 28. 8. 2023.).

[17] S. tim, *systemd-zram-setup@.service generator for zram devices*, 2023. adresa: https://github.com/systemd/zram-generator (pogledano 28. 8. 2023.).

[18] vaeth, *Zram-init*, 2023. adresa: https://github.com/vaeth/zram-init (pogledano 28. 8. 2023.).

[19] atweiden, *Zramen*, 2023. adresa: https://github.com/atweiden/zramen (pogledano 28. 8. 2023.).

[20] G. Alendal, C. Kison i modg, „got HW crypto? On the (in)security of a Self-Encrypting Drive series," *IACR Cryptol. ePrint Arch.*, sv. 2015, str. 1002, 2015. adresa: https://api.semanticscholar.org/CorpusID:10415252.

[21] C. Fruhwirth i c. Org, „New methods in hard disk encryption," kolovoz 2005.

[22] T. Bi Irie guy-cedric, „A Comparative Study on AES 128 BIT AND AES 256 BIT," *INTERNATIONAL JOURNAL OF COMPUTER SCIENCES AND ENGINEERING*, sv. volume 6, str. 30–33, rujan 2018. DOI: 10.26438/ijsrcse/v6i4.3033.

[23] Cloudflare, *Speeding up Linux disk encryption*, 2020. adresa: https://blog.cloudflare.com/speeding-up-linux-disk-encryption (pogledano 28. 8. 2023.).

[24] L. tim, *Filesystem-level encryption (fscrypt)*, 2023. adresa: https://www.kernel.org/doc/Documentation/filesystems/fscrypt.rst (pogledano 28. 8. 2023.).

[25] M. tim, *A guide to mdadm*, 2023. adresa: https://raid.wiki.kernel.org/index.php/A_guide_to_mdadm (pogledano 28. 8. 2023.).

[26] B. tim, *Auto-repair on read*, 2023. adresa: https://btrfs.readthedocs.io/en/latest/Auto-repair.html (pogledano 28. 8. 2023.).

[27] D. I. GmbH, *Btrbk*, 2023. adresa: https://digint.ch/btrbk (pogledano 28. 8. 2023.).

[28] J. Chelladhurai, P. R. Chelliah i S. A. Kumar, „Securing Docker Containers from Denial of Service (DoS) Attacks," *2016 IEEE International Conference on Services Computing (SCC)*, 2016., str. 856–859. DOI: 10.1109/SCC.2016.123.

[29] yosifkit, *Allow arbitrary –user values (mostly) 253*, 2017. adresa: https://github.com/docker-library/postgres/pull/253 (pogledano 28. 8. 2023.).

[30] linuxserver, *Building and maintaining community images*, 2023. adresa: https://www.linuxserver.io (pogledano 28. 8. 2023.).

[31] M. Holt, *The Ultimate Server*, 2023. adresa: https://caddyserver.com (pogledano 28. 8. 2023.).

[32] D. tim, *Docker run reference*, 2023. adresa: https://docs.docker.com/engine/reference/run/#runtime-privilege-and-linux-capabilities (pogledano 28. 8. 2023.).

[33] processone, *eturnal TURN Server*, 2023. adresa: https://eturnal.net/ (pogledano 28. 8. 2023.).

[34] Ezwen, *Problems with large port range binding 3*, 2023. adresa: https://github.com/instrumentisto/coturn-docker-image/issues/3 (pogledano 28. 8. 2023.).

[35] D. tim, *Runtime options with Memory, CPUs, and GPUs*, 2023. adresa: https://docs.docker.com/config/containers/resource_constraints (pogledano 28. 8. 2023.).

[36] D. tim, *Packet filtering and firewalls*, 2023. adresa: https://docs.docker.com/network/packet-filtering-firewalls (pogledano 28. 8. 2023.).

[37] N. tim, *The netfilter.org "iptables" project*, 2023. adresa: https://www.netfilter.org/projects/iptables/index.html (pogledano 28. 8. 2023.).

[38] N. tim, *The netfilter.org "nftables" project*, 2023. adresa: https://www.netfilter.org/projects/nftables/index.html (pogledano 28. 8. 2023.).

[39] N. tim, *IP sets*, 2023. adresa: https://ipset.netfilter.org (pogledano 28. 8. 2023.).

[40] I. tim, *Block Visitors by Country Using Firewall*, 2023. adresa: https://www.ip2location.com/free/visitor-blocker (pogledano 28. 8. 2023.).

[41] N. tim, *The netfilter.org "ulogd" project*, 2023. adresa: https://netfilter.org/projects/ulogd (pogledano 28. 8. 2023.).

[42] D. A. Tevault. 2023.

[43] N. tim, *Nmap: The network mapper*, 2023. adresa: https://nmap.org (pogledano 28. 8. 2023.).

[44] L. kernel tim, *Kernel docs*, 2023. adresa: https://docs.kernel.org/process/1.Intro.html (pogledano 28. 8. 2023.).

[45] DanielMicay, *OS Security: iOS vs GrapheneOS vs stock Android*, 2019. adresa: https://www.reddit.com/r/GrapheneOS/comments/bddq5u/comment/ekxifpa (pogledano 28. 8. 2023.).

[46] D. Vyukov, *The state of the Linux kernel security*, 2020. adresa: https://github.com/ossf/wg-securing-critical-projects/blob/main/presentations/The_state_of_the_Linux_kernel_security.pdf (pogledano 28. 8. 2023.).

[47] A. Murray, *Booting Linux from Linux with kexec*, 2020. adresa: https://www.thegoodpenguin.co.uk/blog/booting-linux-from-linux-with-kexec (pogledano 28. 8. 2023.).

[48] L. kernel tim, *Kernel pointer leak*, 2015. adresa: https://kernsec.org/wiki/index.php/Bug_Classes/Kernel_pointer_leak (pogledano 28. 8. 2023.).

[49] OpenSuse, *Security Hardening: Use of eBPF by unprivileged users has been disabled by default*, 2022. adresa: https://kernsec.org/wiki/index.php/Bug_Classes/Kernel_pointer_leak (pogledano 28. 8. 2023.).

[50] Netflix, *CVE-2019-11477: SACK Panic*, 2019. adresa: https://github.com/Netflix/security-bulletins/blob/master/advisories/third-party/2019-001.md (pogledano 28. 8. 2023.).

[51] N. F. LLC, *The system must be configured to use TCP syncookies when experiencing a TCP SYN flood*, 2015. adresa: https://www.stigviewer.com/stig/oracle_linux_6/2015-06-09/finding/V-50683 (pogledano 28. 8. 2023.).

[52] L. kernel tim, *Yama ptrace$_{s}cope$*, 2023. adresa: https://www.kernel.org/doc/html/latest/admin-guide/LSM/Yama.html (pogledano 28. 8. 2023.).

[53] Kicksecure, *Entropy Sources, Entropy Gathering Daemons, RDRAND*, 2023. adresa: https://www.kicksecure.com/wiki/Dev/Entropy#RDRAND (pogledano 28. 8. 2023.).

[54] L. kernel tim, *Kernel parameters*, 2023. adresa: https://www.kernel.org/doc/Documentation/admin-guide/kernel-parameters.txt (pogledano 28. 8. 2023.).

[55] MITRE, *CWE-416: Use After Free*, 2023. adresa: https://cwe.mitre.org/data/definitions/416.html (pogledano 28. 8. 2023.).

[56] L. kernel tim, *Kernel module signing facility¶*, 2023. adresa: https://www.kernel.org/doc/html/latest/admin-guide/module-signing.html (pogledano 28. 8. 2023.).

[57] P. Enderborg, *[PATCH 2/2] debugfs: Add access restriction option*, 2023. adresa: https://lkml.org/lkml/2020/7/16/122 (pogledano 28. 8. 2023.).

[58] M. Garrett, *Linux kernel lockdown, integrity, and confidentiality*, 2023. adresa: https://mjg59.dreamwidth.org/55105.html (pogledano 28. 8. 2023.).

[59] D. Son, *CVE-2023-2002: RCE flaw in the Bluetooth subsystem of the Linux kernel*, 2023. adresa: https://securityonline.info/cve-2023-2002-rce-flaw-in-the-bluetooth-subsystem-of-the-linux-kernel (pogledano 28. 8. 2023.).

[60] D. tim, *Ddclient*, 2023. adresa: https://ddclient.net/ (pogledano 28. 8. 2023.).

[61] A. Malhotra, I. E. Cohen, E. Brakke i S. Goldberg, „Attacking the Network Time Protocol," 2015., https://eprint.iacr.org/2015/1020. DOI: 10.14722/ndss.2016.23090. adresa: https://eprint.iacr.org/2015/1020.

[62] M. Langer, T. Behn i R. Bermbach, „Securing Unprotected NTP Implementations Using an NTS Daemon," *2019 IEEE International Symposium on Precision Clock Synchronization for Measurement, Control, and Communication (ISPCS)*, 2019., str. 1–6. DOI: 10.1109/ISPCS.2019.8886645.

[63] M. Lichvar, *Chrony*, 2021. adresa: https://chrony-project.org (pogledano 28. 8. 2023.).

[64] G. tim, *Infrastructure - Chrony*, 2023. adresa: https://github.com/GrapheneOS/infrastructure/blob/main/chrony.conf (pogledano 28. 8. 2023.).

[65] S. McIntyre, *Fake Hwclock*, 2022. adresa: https://tracker.debian.org/pkg/fake-hwclock (pogledano 28. 8. 2023.).

[66] CVEdetails, *Security Vulnerabilities (Memory corruption)*, 2023. adresa: https://www.cvedetails.com/vulnerability-list/opmemc-1/memory-corruption.html (pogledano 28. 8. 2023.).

[67] G. tim, *Hardened malloc*, 2023. adresa: https://github.com/GrapheneOS/hardened_malloc (pogledano 28. 8. 2023.).

Popis slika

Popis tablica