ZAVOD ZA ELEKTRONIKU, MIKROELEKTRONIKU, RAČUNALNE I INTELIGENTNE SUSTAVE
FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA
SVEUČILIŠTE U ZAGREBU

DIPLOMSKI RAD br. 1672

Analiza sigurnosti slika diskova za alat Docker

Nikola Kupec

Zagreb, lipanj 2018.

Zahvaljujem se:

- *svom mentoru, doc. dr. sc. Stjepanu Grošu, na strpljenju i pomoći pri izradi ovog rada*
- *svom pjevačkom zboru, Concordiji discors, na osobnoj podršci kroz čitav studij*
- *svojim roditeljima, koji su sa mnom proživljavali brige studiranja*

Moje uspješno okončanje studija i ovaj rad ne bi bili mogući bez njih.

Sažetak

Ovaj se rad bavi istraživanjem sigurnosti alata Docker putem izrade modela prijetnji te izradom programskog alata za automatiziranu provjeru sigurnosti alata Docker.

Abstract

This diploma thesis deals with Docker security research by way of threat modeling, as well as creating a software solution for automatic testing of Docker security concerns.

Sadržaj

1. Uvod

U svijetu računalnih sustava *kontejnerizacija* (engl. *containerization*) postaje ozbiljna alternativa virtualnim strojevima kao način izoliranog i ponovljivog pokretanja i održavanja programske podrške. Mnoge su organizacije, od Microsofta [1] i Googlea [2] do manjih poduzeća [3], prigrlile kontejnerizaciju. Najčešći razlozi za korištenje kontejnerizacije su lakoća razvoja, testiranja i puštanja u produkciju aplikacija. Takozvani *mikroservisi* (engl. *microservices*) posebice su popularna primjena za kontejnerizaciju [4]. Od svih postojećih implementacija kontejnerizacije, Docker je daleko najpoznatija i najkorištenija [3].

Zbog velike popularnosti i lakoće uporabe mnogi su odlučili koristiti Docker, no neki nisu u potpunosti upoznati s mogućim sigurnosnim posljedicama na njihove informacijske sustave [5] [6]. Zbog potrebe za automatiziranim načinom provjere sigurnosti postavljenih Docker poslužitelja tvrtka Docker objavila je vlastiti programski alat za tu svrhu [7].

Ovaj se rad bavi istraživanjem sigurnosti alata Docker izradom modela prijetnji te implementacijom programskog alata za automatsku provjeru sigurnosti alata Docker po uzoru na službeni alat [7]. Za razliku od originalnog, službenog, alata, ova implementacija pisana je u jeziku Python i omogućuje vrlo lako dodavanje i proširivanje provjera sigurnosti. Na putu ka tom cilju ovaj rad istražuje i trenutno stanje modernog modeliranja prijetnji te mjesto alata Docker među ostalim vrstama izolacije programske potpore.

Rad je podijeljen na sedam poglavlja, od kojih je prvo uvodno, a sedmo zaključno. Drugo, treće i četvrto poglavlje posvećeni su teorijskoj razradi, dok se u petom i šestom poglavlju nalazi opis praktičnih pothvata.

Teorijska razrada počinje u drugom poglavlju s istraživanjem i objašnjenjem općenitih pojmova povezanih s izolacijom programske potpore. Potom se teorijska razrada nastavlja u trećem poglavlju s opisima i usporedbama različitih metoda izolacije programske potpore na razini operacijskog sustava – s posebnom pažnjom posvećenom operacijskom sustavu Linux. Konačno, teorijska razrada završava u četvrtom poglavlju gdje se istražuju i opisuju modeliranje prijetnji te neki postupci povezani s modeliranjem prijetnji, kao što su analiziranje aplikacija i prepoznavanje prijetnji u aplikacijama.

Praktični dio rada počinje u petom poglavlju gdje je opisan model prijetnji alata Docker pomoću dijagrama protoka podataka. Nakon opisa modela prijetnji slijedi šesto poglavlje u kojem se nalazi opis programske potpore koja služi provjeri sigurnosti alata Docker, uz mogućnost lakog dodavanja novih provjera.

2. Izolacija programske potpore

Za svakodnevni rad suvremeni informacijski sustavi gotovo se uvijek oslanjaju na programsku potporu – laički rečeno: pokretanje programa na računalima. Nažalost, svaki program koji se izvodi na računalu predstavlja potencijalnu ranjivost. To je zato što, kako bi bio koristan, program mora moći pristupiti resursima računala i / ili *operacijskog sustava* (engl. *Operating System, OS*) kao što su pohrana podataka ili mrežna sučelja. Bez obzira na to proizlaze li ranjivosti iz grešaka u programskoj potpori ili iz nepravilno instalirane ili postavljene programske potpore, to je rizik koji će uvijek postojati. Jedan način ublažavanja takvih ranjivosti jest *izolacija programske potpore* (engl. *software isolation*). Dodatni razlog korištenja izolacije programske potpore jest praktičnost. Ona olakšava i pojednostavljuje postavljanje i održavanje sustava s više korisnika. Programske potpore koja zahtijevaju različite *zavisnosti* (engl. *dependencies*) ili čak različite inačice ili postavke iste zavisnosti mogu sukladno koegzistirati na istom računalnom sustavu bez potrebe da ometaju jedna drugu.

Slijede neki osnovni pojmovi koji bi mogli biti povezani s izolacijom programske potpore i podjela prema njihovim razlikama.

2.1. Virtualizacija

Virtualizacija se danas uglavnom spominje u kontekstu *virtualnih strojeva*, skraćeno VM-ova (engl. *virtual machines*), koji oponašaju cjelokupnu arhitekturu računala te se još naziva i *virtualizacija sklopovlja* (engl. *hardware virtualization*). Ovi VM-ovi su u biti sasvim razdvojeni, s odvojenim operacijskim sustavima, i ponašaju se kao pojedinačni strojevi. Pomalo zbunjujuće, neki autori koriste termine virtualizacije i VM-ova kada se odnose na druge metode izolacije programske potpore [8] [9]. Međutim, ti su izvori stariji od deset godina i svjesno upotrebljavaju općenitije značenje pojmova. Druga, manje srodna upotreba pojma virtualni stroj ima veze s apstrakcijama koje nude programski jezici koji koriste *međukôd* (engl. *byte-code*) za izvođenje, primjerice Java [10] i Python [11], pa čak i jezici koji se izravno prevode u strojni kôd, poput jezika C [12]. Kada se koristi u tim kontekstima pojam VM se odnosi na apstraktne računalne strojeve koji omogućuju programeru razvoj koristeći apstrakcije umjesto konkretnih sklopovskih detalja ovisnih o pojedinom računalu. U ovom se radu uvažava definicija VM-ova koji su povezani s virtualizacijom sklopovlja, što je naizgled najčešće prihvaćena definicija.

2.2. Sandbox

Pojam *zaštićena okolina* (engl. *sandbox*) često se koristi za izolaciju koju pružaju *mrežni preglednik* (engl. *web browser*) [13] i alati za analizu zlonamjernih programa [14] u svrhu izvršavanja nepouzdanog kôda. To je ujedno i definicija koju ovaj rad uvažava. Kao i kod virtualizacije, definicije pojmova se ponekad proturječe ili preklapaju [15], ali opet, izvor je star više od jednog desetljeća i čini se da su se pojmovi stabilizirali od tada.

2.3. Kontejnerizacija

Kontejnerizacija (engl. *containerization*) kao pojam čini se kako je nastao za potrebe opisa izolacije programske potpore na razini OS-a za razliku od virtualizacije koja stvara zasebne VM-ove s vlastitim, zasebnim OS-ovima [16]. Izvori još uvijek ponekad nazivaju kontejnerizaciju oblikom virtualizacije [9]. Kao što je spomenuto na kraju poglavlja 2.1, u ovom radu pojmovi 'virtualizacija' i 'VM' označavat će konkretno virtualizaciju sklopovlja i VM-ove koji sadrže cijele, zasebne operacijske sustave. Kontejneri se razlikuju od VM-ova po tome što koriste metode koje nudi OS i / ili njegova jezgra za ostvarenje izolacije programske potpore. Zbog toga se nazivaju izolacijom na razini OS-a [17].

Ovaj se rad bavi izolacijom programske potpore na razini OS-a.

3. Izolacija programske potpore na razini OS-a

Slijede opisi različitih vrsta izolacije na razini OS-a, počevši od starijih.

3.1. Proces

Najosnovniji oblik podjele rada koji OS omogućuje jest proces [18]. Proces dobiva virtualnu memoriju i (barem jednu) dretvu izvršavanja. On izvršava svoje upute kad mu OS to omogući među ostalim procesima. Proces može koristiti *središnju jedinicu za obradu* (engl. *Central Processing Unit*, *CPU*) za vrlo osnovne operacije kao što su aritmetičke operacije, ali za bilo što složenije mora koristiti *pozive jezgri OS-a* (engl. *system call*). Proces kao koncept također predstavlja izolaciju programske potpore koja je ugrađena u operacijski sustav i može se smatrati rudimentarnom građevnom jedinicom za složenija izolacijska rješenja.

3.2. Naredba chroot

Prvo istinsko rješenje za izolaciju programske potpore vrlo je osnovna naredba *chroot* (dolazi od engleske fraze „change root" [19]), koja je dostupna na svim *POSIX* kompatibilnim operacijskim sustavima, što uključuje *Linux* i *BSD*. Koristeći chroot korisnik može promijeniti *ishodišni direktorij* (engl. *root directory*) procesa, čime se proces ograničava na samo jedan dio datotečnog sustava. Bez *administratorskih prava* (engl. *root privileges*) proces će živjeti u svijetu u kojem je datotečno stablo novog ishodišnog direktorija cijeli datotečni sustav. To ograničeno okružje u kojem se proces tada nalazi naziva se *chroot zatvor* (engl. *chroot jail*) [20].

Međutim, to je opseg mogućnosti chroota. Kako bi se procesu promijenio ishodišni direktorij potrebno je imati administratorske (engl. *root*) ovlasti. Nakon što izvede naredbu chroot korisnik bi se trebao odreći svojih administratorskih prava da osigura da zatvoreni proces ne može izaći van. Kako bi zatvoreni proces mogao funkcionirati unutar chroot zatvora barem neke od *datoteka uređaja* (engl. *device files*) moraju biti dostupne, kao i sve potrebne *biblioteke* (engl. *libraries*) [21]. Budući da se chroot usredotočuje samo na izolaciju datotečnog sustava, zatvoreni program nije ograničen ni na koji drugi način i još uvijek može stupiti u interakciju sa sustavom i potencijalno mu naštetiti. Proces i dalje ima pristup pozivima jezgri OS-a i može komunicirati s drugim procesima. Chroot također ne postavlja nikakva ograničenja na korištenje procesora ili memorijskih resursa.

3.3. FreeBSD Jails

Ovo se rješenje temelji na naredbi chroot, ali po funkcionalnosti ima mnogo veći doseg. *FreeBSD zatvori* (engl. *FreeBSD Jails*) koriste chroot kako bi stvorili mnogo potpuniju izolaciju. Zatvori su prvobitno zamišljeni kao zasebni odjeljci unutar jednog *UNIX* poslužitelja tako da svaki odjeljak ima svoju programsku potporu i njime upravljaju njegovi vlasnici [9]. Osnovni mehanizmi za razdvajanje privilegija i resursa na UNIX sustavima su korisnici, vlasništvo nad datotekama i dozvole. Poul-Henning Kamp smatrao je te osnovne mehanizme neprikladnima za išta više od vrlo jednostavnih potreba. Administracija i održavanje fino zrnatih kontrola pristupa bile su kompliciranе i sklone

pogreškama [22]. Zato je dizajnirao i implementirao FreeBSD zatvore i opisao ih kao lagane virtualne strojeve (engl. *light-weight virtual machines*). Oni nalikuju VM-ovima po tome što programska potpora koja se izvodi ne može lako prepoznati da ne radi u instanci OS-a najviše razine, dok *domaćin* (engl. *host*) može vidjeti unutar sebe, pratiti i izvršavati promjene. Oni nisu kao VM-ovi po tome što se izvode na istoj jezgri OS-a kao i domaćin pa ne trebaju virtualizaciju ili emulaciju sklopovlja.

FreeBSD zatvori unaprijeđenje su u odnosu na naredbu chroot dodavanjem preciznijih kontrola za podešavanje pristupa zatvora datotečnom sustavu, skupu korisnika i mrežnom podsustavu domaćina. Zatvor ima vlastito stablo direktorija, iz kojeg ne može pobjeći, vlastiti *naziv domaćina* (engl. *hostname*) i IP adresu te vlastiti skup korisnika, uključujući i vlastitog root korisnika koji je ograničen na zatvor [23].

Od svih implementacija izolacije programske potpore koje se ne temelje na sklopovskoj virtualizaciji, FreeBSD zatvori bili su prva potpuna implementacija koja je u širokoj uporabi za sigurnost i praktičnost. Kao što im ime govori, dostupni su samo na FreeBSD-u, dok ostali operacijski sustavi imaju konceptualne ekvivalente, kao što su Solaris Zones.

3.4. Solaris Zones

FreeBSD zatvori izravno su nadahnuli razvoj *Solaris zona* (engl. *Solaris Zones*), koje su bile pokušaj poboljšanja zamisli i provedbe FreeBSD zatvora. Solaris zone dizajnirane su tako da budu više integrirane s osnovnim uslugama operacijskog sustava i pružaju više pogodnosti, poput kontrola ograničenja resursa [24].

3.5. Izolacija programske potpore u Linuxu

Dok FreeBSD zatvori, Solaris zone pa čak i naredba chroot predstavljaju gotova izolacijska rješenja, u Linux svijetu postoje mehanizmi koji se koriste u kombinacijama za stvaranje punih rješenja [25].

Linux *prostori imena* (engl. *namespaces*) i *kontrolne grupe* (engl. *control groups, cgroups*) mehanizmi su koje nudi jezgra operacijskog sustava Linux. Pomoću prostora imena moguće je ograničiti što skupine procesa mogu vidjeti, a time i čemu mogu pristupiti, dok kontrolne grupe omogućuju postavljanje ograničenja na procesovo korištenje resursa. *Seccomp* još je jedan mehanizam jezgre. Sa seccompom moguće je zabraniti procesu korištenje određenih poziva jezgri OS-a. Kao što je spomenuto u poglavlju 3.1, proces mora koristiti pozive jezgri OS-a za gotovo bilo kakvu vrstu interakcije sa sustavom, od čitanja ili pisanja datoteke do komuniciranja s drugim procesima [18].

U sljedećim potpoglavljima opisani su neki česti alati i pojmovi u svijetu izolacije programske potpore u Linuxu.

3.5.1. Docker

U poglavlju 3.5 spomenuto je kako su FreeBSD zatvori i Solaris zone različiti od sličnih alata u Linuxu. Zavori i zone su prvorazredni koncepti [25]. S druge strane, u Linuxu, prostori imena i kontrolne grupe su prvorazredni koncepti. Jedan od korisnika prostora imena i kontrolnih grupa je alat Docker, koji je Linux kontejnersko rješenje za kojim ljudi

najčešće posežu. Docker pomoću prostora imena i kontrolnih grupa neizravno koristi primitive jezgre kako bi napravio ono što se naziva Docker kontejnerom.

To u praksi znači da FreeBSD zatvori i Solaris zone dolaze spremni za uporabu i s nekim jamstvima o izolaciji i sigurnosti, ali se ne mogu razbiti u jednostavnije koncepte i ne mogu se upotrijebiti za ono za što nisu bili predviđeni tijekom dizajna. Izgradnja rješenja s Linux primitivima kao građevnim blokovima omogućuje veću fleksibilnost i fino podešavanje, ali unosi veću složenost i potencijalne greške.

Linux kontejneri (engl. *Linux containers*, *LXC*) neko su vrijeme bili podrazumijevani temelj za izvršavanje Dockera, ali već nekoliko godina Docker koristi vlastitu biblioteku naziva libcontainer [26]. Razlozi za ovaj odmak od LXC-a mogu se naći u Dockerovim odgovorima na *često postavljena pitanja* (engl. *frequently asked questions, FAQ*) [27]. Uglavnom se radi o fleksibilnosti i dodatnim olakotnim značajkama koje se dobivaju u odnosu na LXC. Na primjer, LXC ne nudi *prenosivost* (engl. *portability*) kontejnera na različitim računalima domaćinima niti točnu ponovljivost kontejnera. Docker tim značajkama pridaje veliku važnost te je libcontainer dizajniran da ih omogućuje.

U svakom slučaju, LXC i Docker puno se preklapaju u funkcionalnosti te su oba načini izolacije na razini OS-a poznati kao kontejneri [28]. Razlozi za korištenje jednog umjesto drugog nisu uvijek očiti i u mnogim situacijama oba bi vjerojatno dobro funkcionirali. Prema nekom uvriježenom mišljenju [29] čini se da je Docker prikladniji za razvoj, testiranje i pokretanje jednostavnih ponovljivih aplikacija, poput REST web usluga, dok je LXC-ova jača strana stvaranje postojanih kontejnera koji služe kao lagani VM-ovi. Postoje i druga, rjeđe korištena rješenja za kontejnerske sustave kao što su openVZ [30] i rkt [31].

3.5.2. CoreOS

CoreOS je Linux distribucija koja je izgrađena na kontejnerizaciji pomoću Dockera ili, po izboru, rkt. Rkt [31] je vlastita kontejnerska tehnologija CoreOS-a. Zamisao iza CoreOS-a, sada nazvanog *Container Linux*, dok je CoreOS naziv tvrtke, jest imati minimalnu temeljnu distribuciju koja pokreće kontejnerizirane aplikacije koja se lako pokreće na *nakupini* (engl. *cluster*) poslužitelja. CoreOS je oblikovan tako da pomoću njega rukovanje raspodijeljenim sustavom aplikacija bude kao rukovanje jednom jedinicom [32]. Iako se može koristiti na jednom računalu, takvim načinom uporabe gubi se pravi smisao CoreOS-a [33]. Tvrtka CoreOS je u siječnju 2018. najavila da ih kupuje RedHat [34].

3.5.3. Kubernetes

Kubernetes [35] je povezan s raspodijeljenim kontejneriziranim aplikacijama. Nastao je u Googleu te je namijenjen pojednostavljenom stvaranju i upravljanju kontejnerima [36]. Temeljna zamisao Kubernetesa je da se jedna aplikacija sastoji od više kontejnera, od kojih svaki obavlja zasebnu zadaću i zahtjeva različite računalne resurse. Kubernetes olakšava upravljanje takvim aplikacijama grupiranjem kontejnera u logičke jedinice ovisno o aplikacijama koje ti kontejneri sačinjavaju, apstrahirajući složene probleme poput umrežavanja i skaliranja. Kubernetes brzo postaje najpopularniji alat povezan uz *orkestriranje kontejnera* (engl. container *orchestration*) [37]. Orkestriranje kontejnera je automatizacija stavljanja u pogon, upravljanja, skaliranja, umrežavanja i dostupnosti aplikacija građenih na kontejnerima [38]. Kubernetes može upotrebljavati nekoliko

različitih tehnologija kontejnera [39], ali daleko najčešći su Docker kontejneri. Čak je i CoreOS počeo isključivo koristiti Kubernetes za orkestraciju [33].

4. Model prijetnji

4.1. Definicija modela prijetnji

Ne postoji opće prihvaćena definicija pojma m*odeliranje prijetnji* (engl. t*hreat modeling*) [40]. Različiti izvori pružaju različita mišljenja o točnom sadržaju i obujmu postupka modeliranja prijetnji [41] [42], i ponekad koriste nejasne termine [43], no svi se okvirno slažu o cilju – prepoznavanje i rangiranje prijetnji prema informacijskom sustavu [38].

Modeliranje prijetnji kao zamisao o analizi sigurnosti informacijskih sustava ili aplikacija u nekom obliku postoji već desetljećima. Bruce Schneier je 1998. godine u svom radu [44] iznio temeljne zamisli koje su veoma slične današnjim osnovama metoda modeliranja prijetnje. Ukratko, modeliranje prijetnji postupak je koji omogućuje prepoznavanje, pobrojavanje i odmjeravanje potencijalnih prijetnji prema informacijskom sustavu. Postupak modeliranja prijetnji vrši se prvenstveno iz gledišta vanjskoga napadača – „One of the most important lessons to keep in mind when you begin to evaluate a threat model is that your attackers are people, too" [45] – i okvirno se sastoji od tri glavna dijela: prikupljanje informacija o sustavu, određivanje i rangiranje prijetnji, i određivanje eventualnih protumjera [46]. Neki izvori tretiraju rangiranje prijetnji i eventualne protumjere kao postupke zasebne od modeliranja prijetnji [41].

Što se tiče općenite prakse modeliranja prijetnji većina izvora ponavljaju upute navede u Microsoftovoj literaturi [46]. Zapravo, većina izvora, što uključuje predavanja predmeta ZSIS (Zaštita i sigurnost informacijskih sustava) na FER-u, ponavljaju što piše u Microsoftovoj knjizi za sve što ima veze sa modeliranjem prijetnji i koriste grafove i slike preuzete izravno iz te knjige – najčešće bez citiranja. Dok sām Microsoft tu knjigu naziva zastarjelom [46]. Sve što piše u njoj još uvijek može biti valjano, ali nije lako naći službeno očitovanje Microsofta o aktualnom, modernom stanju modeliranja prijetnji.

Uz to, *OWASP* je tijekom izrade ovog rada bio u postupku izmjene svoje dokumentacije o modeliranju prijetnji – primjerice DREAD je potpuno izbačen – „DREAD is DEAD" [47]. OWASP-ove stranice o modeliranju prijetnji su prije također, prilično detaljno, opisivale što se može naći u Microsoftovoj knjizi i koristile slike iz te knjige. Te izmjene radi Adam Shostack, koji je glavni dizajner Microsoftovog SDL-a (engl. *Security Development Lifecycle*) i autor nove knjige o modeliranju prijetnji [48]. Izmijenjene OWASP-ove stranice sada više sliče na tu, novu, knjigu, iako su očigledno još uvijek u postupku izmjene.

4.2. Analiza aplikacije

Prije bavljenja samim prijetnjama u aplikaciji potrebno je analizirati samu aplikaciju i na taj način izraditi model aplikacije. To je nezaobilazan korak koji se ponekad naziva *dekompozicija aplikacije* (engl. *Application Decomposition*) [42]. Tim se korakom nastoji steći razumijevanje o funkcioniranju aplikacije i njezinom međudjelovanju s okolinom. Na taj se način dobiva uvid u *dobra* (engl. *asset*) koja valja štititi, kao i o mogućim gledištima i putanjama potencijalnog napadača.

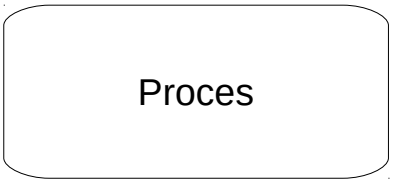
Za obavljanje analize aplikacije, kao i za ostatak modeliranja prijetnji, postoje neke objavljene i prihvaćene metode.

4.2.1. Dijagram protoka podataka

Jedna od najčešćih metoda izrade modela same aplikacije jest *dijagram protoka podataka* (engl. *data flow diagram*, *DFD*) [48]. Problemi često slijede tok podataka i ovim se dijagramom nastoji otkriti te probleme. Dijagram protoka podataka sastoji se od pobrojanih procesa i mjesta pohrane podataka koji se povezuju oznakama protoka podataka i koji međudjeluju sa vanjskim entitetima.

Elementi dijagrama protoka podataka su [48]:

- Proces,
  - Izgled: zaobljeni pravokutnik, kružnica ili višestruka kružnica
    Primjer izgleda u obliku zaobljenog pravokutnika vidljiv je na slici 1.



Slika 1: Element dijagrama protoka podataka koji predstavlja proces

  - Značenje: kôd koji se izvršava
  - Primjer: kôd napisan u programskom jeziku C ili Python
- Tok podataka,
  - Izgled: strelica
    Primjer izgleda je vidljiv na slici 2.

Slika 2: Element dijagrama protoka podataka koji predstavlja tok podataka

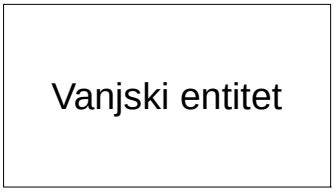
  - Značenje: komunikacija između procesa ili između procesa i pohrane podataka
  - Primjer: mrežne veze
- Pohrana podataka,
  - Izgled: Dvije vodoravne crte između kojih stoji naziv
    Primjer izgleda je vidljiv na slici 3.

Slika 3: Element dijagrama protoka podataka koji predstavlja pohranu podataka

- Značenje: Nešto što pohranjuje podatke
- Primjer: datoteke, baze podataka, komadi dijeljene memorije

• Vanjski entitet

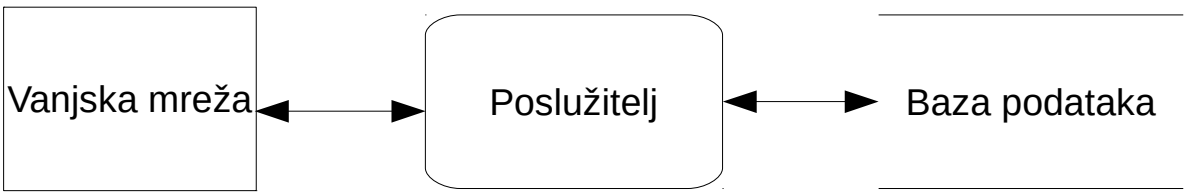
- Izgled: Pravokutnik oštrih kutova
  Primjer izgleda je vidljiv na slici 4.



Slika 4: Element dijagrama protoka podataka koji predstavlja vanjski entitet

- Značenje: Ljudi ili vanjski kôd
- Primjer: Korisnici, klijenti, vanjske usluge ili mrežne stranice

Na slici 5 vidljiv je jedan jednostavan primjer dijagrama protoka podataka u kojem su iskorišteni svi navedeni elementi. U tom je primjeru ilustriran sustav u kojem računalni poslužitelj može zaprimati poruke i zahtjeve od vanjske mreže i slati ih vanjskoj mreži te pohranjivati i čitati podatke u bazi podataka.



Slika 5: Primjer dijagrama podataka

4.2.2. Granice povjerenja

Na bilo koji dijagram modela aplikacije potrebno je dodati granice povjerenja. To su mjesta gdje entiteti koji posjeduju različite razine privilegija međudjeluju. Na dijagramu protoka podataka granica povjerenja izgleda kao isprekidana crta koja presijeca jedan ili više tokova podataka. U iznimnim slučajevima može presijecati i proces ili pohranu podataka, ali je tad potrebno dodatno pojašnjenje.

Granice povjerenja korisne su u modelu aplikacije jer prijetnje teže ka njima. To je zato što granice povjerenja ujedno predstavljaju potencijalnu površinu napada između različitih entiteta u sustavu [48]. Zato valja posvetiti posebnu pažnju granicama povjerenja pri prepoznavanju prijetnji.

4.3. Metode modeliranja prijetnji

U svrhu sistematskog pristupa modeliranju prijetnji stručnjaci su stvorili različite metode koje su danas u uporabi - neke više, neke manje zastupljene. Njihove različitosti proizlaze iz različitosti sustava nad kojima valja vršiti modeliranje prijetnji [49]. Svaki sustav može imati različite potrebe u kontekstu modeliranja prijetnji.

4.3.1. Metoda STRIDE

Od svih metoda modeliranja prijetnji metoda *STRIDE* [50] čini se daleko najpopularnijom budući da je skoro neizostavna u prvim rezultatima web pretrage za pojmom *threat modeling* [42] [51]. Zapravo, metoda STRIDE u tim je primjerima *jedina* spominjana metoda.

Metoda STRIDE razvijena je u Microsoftu i omogućuje određivanje i kategoriziranje prijetnji u nekom sustavu. Njezin je naziv nastao spajanjem početnih slova šest kategorija u koje ona svrstava sve prijetnje, a to su redom:

1. *Zavaravanje* ili *lažiranje* (engl. *Spoofing*)
2. *Zlonamjerna izmjena podataka* (engl. T*ampering*)
3. *Poricanje* (engl. R*epudiation*)
4. *Otkrivanje informacija* (engl. I*nformation disclosure*)
5. *Uskraćivanje usluge* (engl. *Denial of service*)
6. *Povišenje ovlasti* (engl. *Elevation of privilege*)

Ovakva kategorizacija osmišljena je da olakša otkrivanje i popisivanje prijetnji tako da usmjeruje razmišljanje o sustavu i postavljanje pitanja o sigurnosti sustava.

Metoda STRIDE ne bavi se rangiranjem otkrivenih prijetnji, već samo dokumentiranjem i kategoriziranjem, ali usprkos tome neki je izvori poistovjećuju sa *Microsoftovom metodologijom modeliranja prijetnji* [52]. Microsoftova metoda koja se bavi rangiranjem prijetnji i koja uz metodu STRIDE čini svojevrsnu cjelinu naziva se metoda *DREAD*. Iako se te dvije metoda ne preklapaju po namjeni neki ih izvori prikazuju kao konkurentne [53] – ovo ukazuje na neusklađenost i nejasnost u zajednici sigurnosnih stručnjaka po pitanju modeliranja prijetnji.

4.3.2. Metoda DREAD

Metoda DREAD [54], slično metodi STRIDE, dobila je naziv spajanjem početnih slova pet kategorija, no, za razliku od metode STRIDE, metoda DREAD bavi se rangiranjem rizičnosti već prepoznanih prijetnji po tih pet kategorija, koje su redom:

1. *Moguća šteta* (engl. *Damage potential*)
2. *Reproduktivnost* (engl. *Reproducibility*)
3. *Iskoristivost* (engl. *Exploitability*)
4. *Zahvaćeni korisnici* (engl. *Affected users*)
5. *Mogućnost otkrivanja* (engl. *Discoverability*)

Slijedeći ovu metodu svakoj prijetnji daje se ocjena za svaku od pet kategorija, gdje viša ocjena sugerira veću razinu rizika ili opasniju prijetnju. Ocjena može biti broj od 1 do 10 ili jednostavna opisna riječ kao što su „nisko”, „srednje” i „visoko.” Konačna ocjena pojedine prijetnje dobiva se zbrajanjem ocjena svih kategorija i dijeljenjem tog zbroja s pet – drugim riječima, konačna ocjena je aritmetička sredina ocjena svih kategorija. Nakon što se svakoj prijetnji izračuna konačna ocjena dobiva se rang lista rizičnosti prepoznanih prijetnji.

4.3.3. Ostale metode modeliranja prijetnji

Osim već navedenih metoda mrežnom pretragom [49] [55] moguće je naći i neke rjeđe korištene metode:

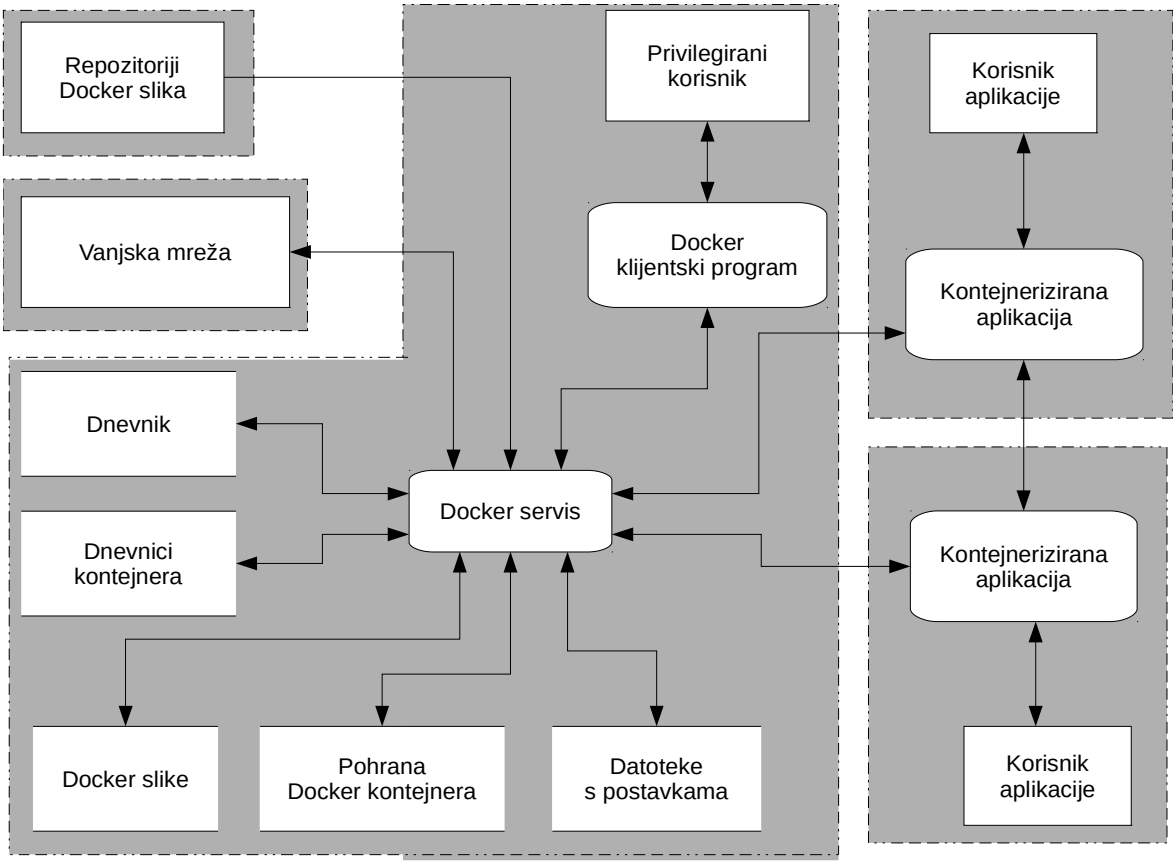
- *PASTA* (*engl. Process for Attack Simulation and Threat Analysis*) [56]
- *Trike* [57]

5. Model prijetnji alata Docker

U ovom poglavlju nalazi se model prijetnji alata Docker. Poglavlje počinje izradom i analizom modela alata Docker i završava prepoznavanjem i kategoriziranjem prijetnji.

5.1. Model alata Docker

Model alata Docker izrađen je u obliku dijagrama protoka podataka, vidljivog na slici 6.



Slika 6: Dijagram protoka podataka alata Docker

5.1.1. Područja povjerenja

Kao što je vidljivo na slici 6, granice povjerenja razgraničuju određeni broj područja povjerenja. Ta područja mogu se podijeliti na tri skupine: servis Docker, kontejnerizirane aplikacije te vanjska mreža i vanjski repozitoriji slika.

5.1.1.1. Servis Docker

Servis Docker (engl. *Docker daemon*) djeluje u jednom zasebnom području povjerenja. U tom se području grade sami kontejneri te se pomoću mehanizama jezgre OS-a uspostavlja izolacija kontejnera. Preko servisa Docker se također pristupa svim informacijama i kontrolama vezanim za kontejnere što uključuje: popisivanje svih kontejnera, pokretanje i zaustavljanje kontejnera te čitanje dnevnika kontejnera.

Za ispravan rad servisu Docker potrebna su administratorska prava, a za korištenje bilo koje funkcionalnosti servisa Docker korisniku je potrebno članstvo u administratorskoj grupi naziva *docker*, ukoliko sām po sebi već nema administratorska prava. Budući da članovi grupe *docker* mogu upravljati servisom Docker, a servis Docker nužno posjeduje administratorska prava, članstvo u grupi *docker* ekvivalentno je administratorskim pravima [58]. Zato se svaki korisnik koji može međudjelovati sa servisom Docker nužno mora smatrati privilegiranim korisnikom.

Najčešći način korištenja funkcionalnosti servisa Docker je putem klijentskog programa. Radi se o programu koji preko *Unix utičnice* (engl. *UNIX socket*), kojoj mogu pristupiti samo administratorski korisnici i korisnici u grupi *docker*, komunicira s *programskim sučeljem* (engl. *application programming interface*, *API*) servisa Docker. Osim Unix utičnice postoji i mrežno sučelje preko protokola *TCP* koje je moguće isključiti. Ako postavke sustava na kojem radi servis Docker ili njegovog mrežnog okružja to na neki način ne brane, moguće je pristupiti mrežnom sučelju servisa Docker iz javne mreže.

Uz sam servis Docker u istom se području povjerenja nalaze i različita mjesta gdje su pohranjeni podaci s kojima rukuje servis Docker i koji služe za pohranu podataka o radu i stanju kontejnera, kao i slika kontejnera.

Za mijenjanje postavki servisa Docker jedna od mogućnosti je datoteka s postavkama. Tim je postavkama moguće podešavati skoro svaki aspekt rada servisa Docker, uključujući korištenje mrežnog sučelja, korištenje *protokola TLS* (engl. *TLS protocol*) uz mrežnu komunikaciju, dopuštanje nepotvrđenih repozitorija Docker slika i slično.

5.1.1.2. Kontejnerizirane aplikacije

Nakon što je izrađen i pokrenut, svaki kontejner ima svoje zasebne procese i korisnike, čak i svog zasebnog administratorskog korisnika (*root* korisnika) koji je ograničen na taj kontejner [58]. Koristeći podrazumijevane postavke, servis Docker kontejnere postavlja u jednu zajedničku virtualnu lokalnu mrežu te ne postavlja ograničenja u njihovoj međusobnoj komunikaciji (engl. *inter-container communication*, *ICC*), niti u njihovoj komunikaciju sa vanjskom mrežom [59].

Prilikom pokretanja novog kontejnera servis Docker može *postaviti* (engl. *mount*) određeni dio domaćinovog datotečnog sustava na raspolaganje korisnicima unutar tog kontejnera. Administrator unutar kontejnera nema nikakvih ograničenja na dostupni dio datotečnog sustava te može učiniti s njim sto god želi.

5.1.1.3. Vanjska mreža i vanjski repozitoriji docker slika

Koristeći podrazumijevane postavke, novoizrađeni kontejneri mogu komunicirati s uređajima na mreži izvan računala domaćina. To je moguće kontrolirati standardnim mrežnim uređajima, kao što su *usmjernici* (engl. *routers)* i *vatrozidovi* (engl. *firewalls*).

Servis Docker može pristupati repozitorijima Docker slika kako bi dohvatio slike za pokretanje novih kontejnera ili kao temelje za izradu novih slika. Ti repozitoriji mogu biti u lokalnoj ili vanjskoj mreži. Koristeći podrazumijevane postavke, servis Docker može slobodno dohvaćati slike koje se nalaze na službenom repozitoriju, *DockerHubu* [60], ili na ispravno postavljenim neslužbenim repozitorijima koji koriste TLS zaštitu mrežnog prometa. Ukoliko repozitorij ne koristi ispravno podešen TLS protokol ili ispravna *vjerodajnica* (engl. *certificate*) nije dostupna servisu Docker tada je potrebno u postavkama navesti taj repozitorij kao nesiguran repozitorij kako bi mu se omogućio nesmetan pristup [61].

5.2. Prepoznavanje i kategoriziranje prijetnji alata Docker

U ovom poglavlju opisano je prepoznavanje i kategoriziranje prijetnji alata Docker pomoću modela iz poglavlja 5.1. U tu svrhu korištena je metoda STRIDE opisana u poglavlju 4.3.1.

Kategoriziranje se radi navođenjem prvog slova engleskog naziva kategorije, a ona su:

1. **S** za *zavaravanje* ili *lažiranje* (engl. *Spoofing*)
2. **T** za *zlonamjernu izmjenu podataka* (engl. *Tampering*)
3. **R** za *poricanje* (engl. *Repudiation*)
4. **I** za *otkrivanje informacija* (engl. *Information disclosure*)
5. **D** za *uskraćivanje usluge* (engl. *Denial of service*)
6. **E** za *povišenje ovlasti* (engl. *Elevation of privilege*)

Slijedi popis prepoznanih prijetnji uz njihove kratke opise i svrstavanje u (barem) jednu od šest kategorija metode STRIDE.

1. *Lažno predstavljanje Docker repozitorija.*

   Ukoliko servis Docker nema ispravno postavljenu provjeru TLS vjerodajnica maliciozni se izvor može lažno predstavljati kao pravi, namijenjeni repozitorij i tom servisu Docker predati maliciozne slike. Servis Docker u tom slučaju ne bi sām po sebi mogao primijetiti zavaravanje i prihvatio bi te slike kao valjane. Ovo predstavlja veliki rizik jer se podmetnute slike tada mogu pokretati uz administratorska prava nad komadom datotečnog sustava domaćina.
   Kategorija u koju ova prijetnja prvenstveno spada jest **S** uz eventualnu kategorizaciju u svih pet ostalih kategorija zbog mogućih velikih ovlasti koje maliciozni, podmetnuti kôd može steći.

2. *Podmetanje lažne izvršne datoteke servisa Docker*.

   Budući da servis Docker mora biti pokrenut s administratorskim pravima kako bi mogao funkcionirati, njegovu izvršnu datoteku često pokreće upravo administratorski korisnik sustava. Ako umjesto prave izvršne datoteke bude podmetnut istoimeni maliciozni program tada takvo pokretanje servisa Docker predstavlja najozbiljnije narušavanje sigurnosti sustava.
   Ova prijetnja prvenstveno je kategorizirana kao **S**, ali može se ujedno kategorizirati i kao **E**, a time i kao sve ostale kategorije metode STRIDE.

3. *Kompromitiranje pohrane Docker kontejnera*.

   Ako napadač uspije pristupiti pohrani Docker kontejnera, gdje su svi svi podaci potrebni za ponovno izvršavanje izrađenih kontejnera, može pristupiti povjerljivim podacima iz tog kontejnera te ih razotkriti, mijenjati ili uništiti. Time se ova prijetnja kategorizira pod **T**, **I** i **D**.

4. *Kompromitiranje dnevnika kontejnera*.

   Dnevnički zapisi o radu pojedinog kontejnera servis Docker zapisuje i daje na uvid privilegiranim korisnicima na njihov zahtjev. Ako zlonamjerna osoba ima pristup tim dnevnicima može ih izbrisati ili izmijeniti. Po uobičajenim postavkama servisa Docker, korisnik mora imati administratorska prava, ili njima ekvivalenta prava, da bi mogao zatražiti dnevničke zapise. Ako ne postoji nikakvo osiguranje da ti dnevnici nisu bili neovlašteno mijenjani, primjerice održavanje kopije na udaljenom poslužitelju, postoji mogućnost da jedan takav korisnik izmijeni dnevnike i ne bude otkriven.
   Ova prijetnja spada pod kategoriju **T** i **R**.

5. *Kontejner čita mrežne pakete ostalih kontejnera*.

   Po podrazumijevanim postavkama svi kontejneri na jednom poslužitelju smješteni su u istu virtualnu lokalnu mrežu. To znači da svaki kontejner može čitati mrežne pakete koje ostali kontejneri međusobno šalju i time otkriti povjerljive informacije.
   Ova prijetnja spada u kategoriju **I**.

6. *Korisnik s niskim ovlastima u kontejneru povisi svoje ovlasti*.

   Podrazumijevane postavke omogućuju korisniku u kontejneru koji posjeduje male ovlasti da povisi svoje ovlasti promjenom svojeg praktičnog UID-a, primjerice izvršavanjem programa kojem je postavljena zastavica *setuid*. Program kojem je postavljena zastavica setuid izvršava se s ovlastima svog vlasnika umjesto korisnika koji ga pokreće.
   Ova prijetnja spada u kategoriju **E**.

Nabrojene prijetnje sažete su u tablici 1. Na desnoj strani tablice unesena su slova koja predstavljaju STRIDE kategorije u koje pojedina prijetnja spada. Prazno polje znači da prijetnja ne spada u tu kategoriju. Podebljano veliko slovo znači da prijetnja spada u tu kategoriju. Malo slovo znači da je smještanje prijetnje u tu kategoriju diskutabilno.

Lažno predstavljanje Docker repozitorija	**S**	t	r	i	d	e
Podmetanje lažne izvršne datoteke servisa Docker	**S**	t	r	i	d	**E**
Kompromitiranje pohrane Docker kontejnera		**T**		**I**	**D**	
Kompromitiranje dnevnika kontejnera		**T**	**R**			
Kontejner čita mrežne pakete ostalih kontejnera				**I**		
Korisnik s niskim ovlastima u kontejneru povisi svoje ovlasti	s	t	r	i	d	**E**

Tablica 1: Sažetci prijetnji

U ovom su poglavlju navedene sve nađene prijetnje u alatu Docker i njihova kategorizacija

po metodi STRIDE. Usmjeravanjem istrage pomoću kategorija metode STRIDE olakšalo se nalaženje te opisivanje prijetnji.

6. Program za analizu sigurnosti alata Docker

U sklopu ovog diplomskog rada izrađena je programska potpora za analizu sigurnosti alata Docker pod nazivom *docker-bench-security-python* [62]. Izrađena je po uzoru na službenu Dockerovu skriptu za UNIX ljusku koja služi za analizu sigurnosti alata Docker pod nazivom „docker-bench-security" [7]. Kao i službena Dockerova skripta, ova programska potpora prvenstveno je namijenjena provjeri sigurnosti Dockera po točkama navedenim u dokumentu *CIS*-a (engl. *Center for Internet Security*, *CIS*) [63]. Cjelokupna provjera Dockera sastoji se od niza pojedinačnih provjera (engl. *checks*). Te provjere su zasebni, relativno mali dijelovi kôda od kojih svaki provjerava jedan aspekt sigurnosti alata Docker.

Program je oblikovan je tako da:

- po uputama u CIS-ovom dokumentu i po uzoru na Dockerovu službenu skriptu:
  - provjere sigurnosti mogu biti bodovane,
  - određene provjere mogu biti samo informativne, bez ikakvog bodovanja
- pored uputa u CIS-ovom dokumentu i Dockerove službene skripte:
  - nove provjere mogu biti dodane već postojećim provjerama,
  - provjere mogu biti grupirane u skupove,
  - dodavanje nove provjere zahtjeva samo dodavanje jedne funkcije ili modula u direktorij gdje se nalaze provjere,
  - moguće je pokretati samo podskup svih provjera, umjesto sve provjere,
  - postoje pomoćni razredi i funkcije koji olakšavaju i ubrzavaju izradu novih provjera, a među njima su:
    - funkcije za dohvaćanje procesa i rukovanje procesima na poslužitelju,
    - funkcije za provjeru postavki *Linux Audit* alata [64] i povezanih alata Linux sustava,
    - funkcije za dohvaćanje mogućih postavki alata Docker i njihovih podrazumijevanih vrijednosti, u obliku postavki u naredbenom retku i datoteke s postavkama,
    - razred za dohvaćanje postavljenih i / ili važećih postavki alata Docker,
    - funkcije za dohvaćanje procesa koji pripada alatu Docker,
  - svi dijelovi kôda imaju svoje *jedinične testove* (engl. *unit tests*)

Implementacijske pojedinosti programa uključuju:

- kôd je napisan u verziji 3 programskog jezika *Python*,
- nazivi funkcija, varijabli i razreda, kao i svi komentari i opisi u kôdu pisani su na engleskom jeziku,
- nastojalo se da što je moguće veći dio kôda bude opisan jasnim komentarima,

- kôd je pisan tako da skoro uvijek bude manje od 81 znak po retku i da uvijek bude manje od 121 znak po retku,
- korišten je *sustav za verzioniranje kôda* (engl. *version control system*, *VCS*) *Git*,
- kod svake *predaje* (engl. *commit*) u sustav Git nastojalo se da što je moguće manji i konzistentniji komad kôda bude pohranjen i da pohranjivanje bude popraćeno sažetim i točnim opisom,
- Git repozitorij programa postavljen je na stranicu GitHub [62].

6.1. Arhitektura programa

Ispis 1 prikazuje strukturu vidljivu u vršnom direktoriju programa.

```
docker-bench-security-python/
├── checks
├── docker-bench-security.py
├── helpers
├── Pipfile
├── Pipfile.lock
├── README.md
└── tests
    ├── checks
    └── helpers
```

Ispis 1: Ispis naredbe tree u vršnom direktoriju programa

Kao što je vidljivo u ispisu 1, kôd je strukturiran na jedan prilično jednostavan način. U vršnom se direktoriju nalaze tri ključna direktorija:

- *checks* – u kojem su same provjere koje provjeravaju sigurnost alata Docker i razredi koje provjere **moraju** izravno upotrebljavati
- *helpers* – u kojem su pomoćne funkcije i razredi kojima se provjere mogu koristiti za mnogo lakše i brže provjeravanje
- *tests* – u kojem su jedinični testovi kôda, raspoređeni u pripadajuće poddirektorije ovisno o tome koji komad kôda testiraju – na primjer, testovi za kôd iz *helpers* direktorija nalaze se u poddirektoriju *tests/helpers*

U vršnom se direktoriju također nalaze sljedeće datoteke:

- *docker-bench-security.py* – koja je točka pokretanja za čitavu programsku potporu i u kojoj se nalazi razred koji posjeduje vršne funkcionalnosti programske potpore, drugim riječima: određivanje provjera za pokretanje, pokretanje provjera, dobavljanje rezultata, računanje bodova i slično
- *Pipfile* i *Pipfile.lock* – koje služe kao nova zamjena za stariji *requirements.txt* format zapisivanja programskih zahtjeva *Python paketa* (engl. *Python package*) [65]

Slijede potpoglavlja s detaljnijim opisima pojedinih komada kôda.

6.1.1. Checks

Poddirektorij *checks* osmišljen je i implementiran tako da se u njemu može naći sve što je neophodno za izradu novih provjera. U ispisu 2 vidljiva je struktura poddirektorija *checks*.

```
├── checks
    ├── __init__.py
    ├── checks_1_host_configuration.py
    ├── checks_2_docker_daemon_configuration.py
    └── result.py
```

Ispis 2: Ispis naredbe tree u poddirektoriju checks

6.1.1.1. Modul __init__.py

U modulu *__init__* nalazi se kôd koji služi za implementaciju, označavanje i prikupljanje provjera. Konkretno, to su razred *Check* i *dekorator funkcija* (engl. *function decorator*) *check*. Kada se pokrene, kôd u ovom modulu pretraži direktorij *checks* za svim funkcijama koje koriste dekorator *check*. Za svaku tako nađenu funkciju on tada stvara novi objekt razreda *Check* i tu funkciju pohranjuje kao funkciju provjere u tom objektu.

Svaki objekt razreda *Check* ima metodu zvanu *run* (hrv. pokreni) s kojom se provjera pokreće i koja vraća rezultat provjere u obliku objekta razreda *Result*. Objekti razreda *Check* također imaju metodu zvanu *evaluate* (hrv. ocjeni) koja vraća ocjenu rezultata provjere.

Dekorator *check* obilježava funkciju kao funkciju provjere, u kojoj je implementirana sāma provjera. Dekorator *check* može se po želji pozivati sa parametrom *weight* (hrv. težina), koja se upotrebljava kasnije prilikom ocjenjivanja rezultata provjera. Parametar *weight* se prilikom ocjenjivanja množi sa dobivenom ocjenom provjere, a njegova podrazumijevana vrijednost je 1,0 (jedan). Postavljajući parametar *weight* na vrijednost 0,0 (nula) moguće je označiti provjeru kao neocjenjivanom.

Uobičajeni način za dohvaćanje svih provjera pomoću paketa *checks* je sljedeći kôd:

```
from checks import CHECK_MODULES
```

CHECK_MODULES je ovdje lista koja sadrži sve nađene provjere, podijeljene po modulu u kojem se nalaze. Slijedi primjer:

U direktoriju *checks* postoje dva modula, naziva *provjere_1* i *provjere_2*, te se u modulu *provjere_1* nalaze funkcije *provjera_1_1* i *provjera_1_2*, a u modulu *provjere_2* se nalazi funkcija *provjera_2_1*. Uvezena lista *CHECK_MODULES* sadrži dva objekta tipa *CheckModule*. Oba imaju atribut naziva *checks*, koji je lista. U prvom ta lista sadrži provjere *provjera_1_1* i *provjera_1_2*, a u drugom ona sadrži samo provjeru *provjera_2_1*.

Modul *result* sadrži implementaciju razreda *Result* i svih mogućih rezultata provjera, od kojih su najvažniji *PASSED*, kada je provjera uspješno položena, i *FAILED*, kada provjera

nije uspješno položena. Svaki rezultat, u obliku objekta razreda *Result*, može također sadržavati poruku ili drugačije detalje o provjeri i njezinom izvršavanju.

6.1.1.2. Provjere sigurnosti alata Docker

Provjere, u obliku funkcija, moraju zadovoljavati neke kriterije da bi bile uspješno pronađene te pokrenute i ocijenjene:

- Svaka funkcija provjere mora koristiti dekorator *check*.
- Naziv funkcije je proizvoljan.
- Funkcija ne smije imati vlastite parametre.
- *Dokumentacijski tekst* (engl. *docstring*) na početku funkcije služi za opisivanje provjere i koristi se kod prikaza rezultata i ocjena provjera.
- Svaka funkcija provjere mora vratiti konačan status provjere u obliku enumeracijskog objekta Status, definiranog u modulu *result*, ili vrijednosti istinitosti.
- Za jednostavnije implementiranje funkcija provjera vrijednosti istinitosti *True* i *False* ekvivalentne su enumeracijskim objektima *PASSED* i *FAILED*.

Oblik funkcije ilustriran je pseudo-kôdom u ispisu 3.

```
@check(weight=1.0)
def prva_provjera():
    """Ovdje je mjesto opisu provjere. Može navoditi što
       provjera provjerava."""

    if broj_koji_je_veci_od_dva < 2:
        return FAILED, """Provjera nije položena jer se
                          broj pokazao manjim od dva"""

    if pomocna_funkcija() == 'Tekst':
        return PASSED, 'Provjera je uspješna jer ...'
```

Ispis 3: Oblik funkcije provjere

U modulima *checks_1_host_configuration* i *checks_2_docker_daemon_configuration* nalaze se primjeri implementacija provjera. Jedan takav primjer vidljiv je u ispisu 4.

U primjeru u ispisu 4 vidljivo je korištenje dekoratora *check* bez navedenog parametra, u kojem slučaju parametar *weight* dobiva podrazumijevanu vrijednost 1,0 (jedan). Funkcija nema vlastitih parametara jer bi postavljanje parametara u definiciji funkcije provjere izazvalo greške prilikom kasnijeg dohvaćanja provjera. Ova funkcija vraća vrijednosti *True* ili *False* umjesto eksplicitnih enumeracijskih objekata.

```
@check()
def check_1_6():
    '''1.6  - Ensure auditing is configured for Docker
files and directories - /var/lib/docker'''

    if not os.path.isdir('/var/lib/docker'):
        return False

    if not auditctl_installed():
        return False

    if path_in_current_audit_rules('/var/lib/docker'):
        return True

    if path_in_audit_rules_file('/var/lib/docker'):
        return True
```

Ispis 4: Primjer funkcije provjere

Kao funkcije provjera omogućeno je i korištenje *generatorskih funkcija* (engl. *generator functions*) umjesto običnih funkcija. Na taj se način omogućuje da provjera bude uspješna u jednom dijelu, ali ne u potpunosti te da se pritom vrate informacije o tome gdje i zašto provjera nije uspjela. Primjer takve provjere vidljiv je u ispisu 5.

```
@check()
def check_2_12():
    '''2.12 - Ensure centralized and remote logging is
       configured'''

    if options['log-driver'] != 'json-file':
        yield PASSED, 'PASS by effective option value'
    else:
        yield FAILED, '''WARN log-driver set to "json-
                         file"'''

    client = docker.from_env()
    docker_info = client.info()

    if docker_info['LoggingDriver'] != 'json-file':
        yield PASSED, '''PASS by current "docker info"
                         output'''
    else:
       yield FAILED, '''WARN LoggingDriver is set to
                        "json-file" in current "docker
                        info" output'''
```

Ispis 5: Primjer generatorske funkcije provjere

U primjeru u ispisu 5 vidi se korištenje ključne riječi u programskom jeziku Python zvane *yield*. Ta ključna riječ vraća vrijednost, ali, za razliku od ključne riječi *return*, ostavlja mogućnost da funkcija kasnije nastavi izvođenje od tog mjesta. Funkcije koje koriste ključnu riječ *yield* umjesto ključne riječi *return* smatraju se generatorskim funkcijama. Koristeći generatorske funkcije za funkcije provjera moguće je držati različite dijelove provjere i njihove logičke mehanizme odvojenima. Dekorator *check* kasnije se pobrine da ovakve funkcije budu ispravno izvršene i da njihov rezultat zapravo bude lista objekata razreda *Result* koja ima svoj način ocjenjivanja.

U ovom je primjeru također vidljiv pomoćni objekt *options*, koji bitno olakšava provjere vezane uz postavke servisa Docker. Taj pomoćni objekt proizlazi iz razreda *Options*, koji je opisan u potpoglavlju 6.1.2.1.

6.1.2. Helpers

U poddirektoriju *helpers* nalaze se pomoćne funkcije i razredi koji su vezani za samo provođenje provjera i provjeravanje njihovih uvjeta. U ispisu 6 vidljiva je struktura poddirektorija *helpers*.

```
├── helpers
    ├── __init__.py
    ├── docker_daemon_options.py
    ├── docker_daemon.py
    ├── dockerd_doc.py
    ├── exceptions.py
    ├── linux_audit.py
    └── processes.py
```

Ispis 6: Ispis naredbe tree u poddirektoriju helpers

6.1.2.1. Modul docker_daemon_options

Ovaj se modul bavi olakšavanjem dohvaćanja postavki servisa Docker. Njegovo je javno sučelje razred *Options*. Objekti razreda *Options* sadrže sve informacije vezane uz podrazumijevane, izričito postavljene i konačno korištene postavke servisa Docker i to postižu koristeći funkcionalnosti različitih ostalih pomoćnih funkcija i razreda u direktoriju *helpers*.

Postoje dva načina definiranja postavki za servis Docker [61]:

1. Korištenje zastavica i parametara prilikom pokretanja servisa Docker u naredbenom retku
2. Postavljanje vrijednosti u datoteci s postavkama servisa Docker, koja je na Linuxu najčešće */etc/docker/daemon.json*

Konačne vrijednosti postavki određuju se po sljedećem redu, pri čemu naredbeni redak ima najviši prioritet, a podrazumijevane vrijednosti najniži:

1. Naredbeni redak
2. Datoteka s postavkama
3. Podrazumijevane vrijednosti

Razred *Options* prvo saznaje sve moguće postavke u naredbenom retku i u datoteci s postavkama i sve podrazumijevane vrijednosti. Zatim dohvaća sve izričito postavljene postavke u naredbenom retku i datoteci s postavkama te određuje koje su vrijednosti konačne, važeće po navedem poretku.

Za otkrivanje sadržaja naredbenog retka pokrenutog servisa Docker razred *Options* koristi pomoć modula *docker_deamon*, koji je opisan u potpoglavlju 6.1.2.2, a za pomoć pri otkrivanju podrazumijevanih vrijednosti i iščitavanje važećih postavki iz naredbenog retka koristi modul *dockerd_doc*, koji je opisan u potpoglavlju 6.1.2.3.

6.1.2.2. Modul docker_daemon

Modul *docker_daemon* prilično je sažet. On sadrži samo tri pomoćne funkcije:

1. Funkcija kojom se otkriva proces u kojem se trenutno izvršava servis Docker
2. Funkcija koja vraća sadržaj naredbenog retka trenutnog procesa servisa Docker

3. Funkcija koja provjerava može li korisnik pristupiti servisu Docker

Ove funkcije su prvenstveno napravljene da pomognu pri otkrivanju važećih postavki servisa Docker.

Za pretraživanje svih procesa kako bi se našao proces u kojem se izvršava servis Docker upotrebljava se modul *processes*, opisan u potpoglavlju 6.1.2.3 sa preostalim modulima.

6.1.2.3. Preostali moduli poddirektorija helpers

U poddirektoriju *helpers* nalaze se još nekoliko manjih modula čije su funkcionalnosti dovoljno specifične da budu izdvojene. Ti moduli su:

- *Modul dockerd_doc*.

  On sadrži samo nekoliko konstanti povezanih sa dokumentacijom servisa Docker:

  - Tekst službene dokumentacije naredbenog retka servisa Docker.
  - Tekst služenog primjera datoteke s postavkama servisa Docker.

  Te su tekstualne konstante potrebne za otkrivanje podrazumijevanih vrijednosti i za iščitavanje vrijednosti iz naredbenog retka servisa Docker te za iščitavanje postavljenih vrijednosti u datoteci s postavkama.

- *Modul exceptions.*

  Ovo je samo modul namijenjen za sadržavanje svih iznimki (engl. exceptions) koje se koriste u ostatku programa.

- *Modul linux_audit*.

  Za neke provjere potrebno je ispitati kako je Linuxov *audit servis* postavljen. Ovaj modul sadrži pomoćne funkcije za provjeru dostupnosti audit servisa i postoje li određeni zapisi u postavkama audit servisa.

- *Modul processes*.

  U ovom se modulu nalaze pomoćne funkcije za otkrivanje procesa koji se trenutno izvršavaju. Za rad ovog modula ključna je vrlo korisna i odlična biblioteka *psutil* [66], bez koje bi pretraživanje procesa u Pythonu bilo mnogo teže.

6.1.3. Modul docker-bench-security

Ovaj je Python modul glavna točka pokretanja za cijeli program. Sadrži ključni razred *DockerBenchSecurity* u kojem su sučelja prema glavnim funkcionalnostima potrebnim za postavljanje, izvršavanje i ispisivanje cijelog programa:

1. Koristi *checks* paket za dohvaćanje svih mogućih provjera
2. Ovisno o odabiru korisnika uzima podskup dohvaćenih provjera
3. Pokreće odabrane provjere
4. Ocjenjuje odabrane provjere
5. Ispisuje statuse odabranih provjera, njihove ocjene i ukupno bodovanje

U ispisu 7 vidljiv je primjer rezultata izvršavanja modula docker-bench-security. Zbog velikog broja redaka prikazan je samo kraj ispisa, koji je dovoljan da se stekne dojam o ispisu programa.

```
[FAILED - WARN live-restore not set to True, FAILED - WARN LiveRestoreEnabled is not
set to True in current "docker info" output] : 0 : check_2_14 : 2.14 - Ensure live
restore is Enabled
-------------------------------------------------------------------------------
FAILED - WARN userland-proxy not set to false : 0 : check_2_15 : 2.15 - Ensure
Userland Proxy is Disabled
-------------------------------------------------------------------------------
PASSED - PASS by current "docker info" output : 0 : check_2_16 : 2.16 - Ensure
daemon-wide custom seccomp profile is applied, if needed
-------------------------------------------------------------------------------
[PASSED - PASS, PASSED - PASS by current "docker info" output] : 2 : check_2_17 :
2.17 - Ensure experimental features are avoided in production
-------------------------------------------------------------------------------
FAILED - WARN no-new-privileges not set to true : 0 : check_2_18 : 2.18 - Ensure
containers are restricted from acquiring new privileges
-------------------------------------------------------------------------------
PASSED - None : 1 : check_2_2 : 2.2  - Ensure the logging level is set to 'info';
-------------------------------------------------------------------------------
PASSED - None : 1 : check_2_3 : 2.3  - Ensure Docker is allowed to make changes to
iptables
-------------------------------------------------------------------------------
PASSED - None : 1 : check_2_4 : 2.4  - Ensure insecure registries are not used
-------------------------------------------------------------------------------
PASSED - None : 1 : check_2_5 : 2.5  - Ensure aufs storage driver is not used
-------------------------------------------------------------------------------
PASSED - INFO Not listening on TCP : 1 : check_2_6 : 2.6  - Ensure TLS authentication
for Docker daemon is configured
-------------------------------------------------------------------------------
FAILED - WARN default-ulimit not set : 0 : check_2_7 : 2.7  - Ensure the default
ulimit is configured appropriately
-------------------------------------------------------------------------------
FAILED - WARN userns-remap not set : 0 : check_2_8 : 2.8  - Enable user namespace
support
-------------------------------------------------------------------------------
PASSED - PASS : 1 : check_2_9 : 2.9  - Ensure the default cgroup usage has been
confirmed
-------------------------------------------------------------------------------

Final score:
11.0 / 28.0
```

Ispis 7: Izvršavanje programa docker-bench-security.py

7. Zaključak

Čini se da se današnji način održavanja i upravljanja računalnih sustava nezaustavljivo kreće prema kontejnerizacijskim tehnologijama. Kontejneri kao neka vrsta malih virtualnih strojeva postoje već desetljećima i tek su relativno nedavno postali veoma popularni te se njihov nagli uspon može pripisati usvajanju kontejnera od strane velikih korporacija, pri čemu se Docker marketinški našao u najboljem položaju. Kontejneri i Docker danas su skoro sinonimi, no ne postoji neki tehnički uvjerljiv razlog zašto druge kontejnerske tehnologije nisu također popularne. Bez obzira na tehničke prednosti i mane Docker će zasigurno nastaviti biti na prvom mjestu jer polako postaje standard koji se očekuje. Zbog toga je znanje o Dockeru mnogo više prenosivo između različitih radnih mjesta i situacija.

Još jedna prednost koju Docker ima je opširnost i količina informacija i materijala za učenje - službenih i neslužbenih. Odluka tvrtke Docker da objavi svoj kôd za provjeru sigurnosti pod slobodnom licencom omogućila mi je da dublje istražujem o sigurnosti Dockera i da po tom uzoru izradim svoju, napredniju, implementaciju.

Znanost modeliranja prijetnji čini se istovremeno stara, čvrsto utemeljena i nova, tek u povojima. Mnogi izvori zdravo za gotovo navode činjenice bez citata. Ponavljane metode čine se kao da su oduvijek postojale. Izvor skoro svih konkretnih materijala o modeliranju prijetnji čini se Microsoft, koji se polako kreće u svom smjeru. Tehnologije i metodologije nikad se ne zaustavljaju već se razvijaju i moje je mišljenje da bi bilo dobro kad bi svijet modeliranja prijetnji malo dublje pogledao u svoje temelje i odlučio izvući neke osnovne točke i termine oko kojih se većina može složiti i od kojih je daljnji razvoj moguć. Srećom, izgleda da se OWASP, uz pomoć autora Adama Shostacka, kreće u tom smjeru. No možda je modeliranje prijetnji samo po sebi presubjektivno i ovisno o pojedinoj situaciji da bi se moglo čvrsto definirati i prizemljiti.

Literatura

[1] S. Fulton, „Why Containers Are Everywhere in Microsoft Azure“, *The New Stack*, 25-ruj-2017. [Na internetu]. Dostupno na: https://thenewstack.io/attracts-developers-microsoft-azure-app-service-linux/. [Pristupljeno: 27-lip-2018].

[2] „What are Containers and their benefits“, *Google Cloud*. [Na internetu]. Dostupno na: https://cloud.google.com/containers/. [Pristupljeno: 27-lip-2018].

[3] „8 surprising facts about real Docker adoption“, *Datadog Infrastructure Monitoring*, lip-2018. [Na internetu]. Dostupno na: https://www.datadoghq.com/docker-adoption/. [Pristupljeno: 27-lip-2018].

[4] J. Julien, „State of the Union of Microservices and Containers“, *Stackify*, 27-velj-2018. [Na internetu]. Dostupno na: https://stackify.com/microservices-containers/. [Pristupljeno: 27-lip-2018].

[5] D. Gewirtz, „Weak Docker security could lead to magnified vulnerabilities due to efficiency of containers“, *ZDNet*, 24-srp-2017. [Na internetu]. Dostupno na: https://www.zdnet.com/article/weak-docker-security-could-lead-to-magnified-cybersecurity-threat-due-to-efficiency-of-containers/. [Pristupljeno: 27-lip-2018].

[6] J. Oltsik, „Containers are here. What about container security?“, *CSO Online*, 15-svi-2018. [Na internetu]. Dostupno na: https://www.csoonline.com/article/3273347/security/containers-are-here-what-about-container-security.html. [Pristupljeno: 27-lip-2018].

[7] „docker-bench-security“, 17-lip-2018. [Na internetu]. Dostupno na: https://github.com/docker/docker-bench-security. [Pristupljeno: 17-lip-2018].

[8] A. Singh, „An Introduction to Virtualization“, *kernelthread.com*, sij-2004. [Na internetu]. Dostupno na: http://www.kernelthread.com/publications/virtualization/. [Pristupljeno: 14-lip-2018].

[9] P.-H. Kamp, „Jails – High value but shitty Virtualization —“, *PHKs Bikeshed*. [Na internetu]. Dostupno na: http://phk.freebsd.dk/sagas/jails.html. [Pristupljeno: 14-lip-2018].

[10] B. Venners, „Java Virtual Machine’s Internal Architecture“. [Na internetu]. Dostupno na: https://www.artima.com/insidejvm/ed2/jvm.html. [Pristupljeno: 14-lip-2018].

[11] R. Pattis, „Week Nine: Executing Code in The Python Virtual Machine“, *Enhanced ICS 33 Notes*. [Na internetu]. Dostupno na: https://www.ics.uci.edu/~brgallar/week9_3.html. [Pristupljeno: 14-lip-2018].

[12] „No, I am referring to the virtual machine defined by the C language.“, *Hacker News*, 10-sij-2013. [Na internetu]. Dostupno na: https://news.ycombinator.com/item?id=5038048. [Pristupljeno: 14-lip-2018].

[13] „Sandbox“, *Chromium Source Docs*. [Na internetu]. Dostupno na: https://chromium.googlesource.com/chromium/src/+/master/docs/design/sandbox.md. [Pristupljeno: 14-lip-2018].

[14] „Cuckoo Sandbox - Automated Malware Analysis“. [Na internetu]. Dostupno na: https://cuckoosandbox.org/. [Pristupljeno: 14-lip-2018].

[15] V. Prevelakis i D. Spinellis, „Sandboxing Applications“, str. 8.

[16] P. Rubens, „The Benefits of Docker vs. Server Virtualization“, *ServerWatch*, 19-kol-2014. [Na internetu]. Dostupno na: https://www.serverwatch.com/server-trends/the-benefits-of-docker-vs.-server-virtualization.html. [Pristupljeno: 14-lip-2018].

[17] J. Topjian, „Contain your enthusiasm – Part One: a history of operating system containers“, *Cybera*, 12-stu-2013. [Na internetu]. Dostupno na: http://www.cybera.ca/news-and-events/tech-radar/contain-your-enthusiasm-part-one-a-history-of-operating-system-containers/. [Pristupljeno: 14-lip-2018].
[18] N. Mavrogiannopoulos, „Software Isolation in Linux“, *nmav's Blog*, 15-lip-2015. [Na internetu]. Dostupno na: https://nikmav.blogspot.com/2015/06/software-isolation-in-linux_15.html. [Pristupljeno: 14-lip-2018].
[19] „chroot(8)“, *FreeBSD Manual Pages*. [Na internetu]. Dostupno na: https://www.freebsd.org/cgi/man.cgi?query=chroot&sektion=8. [Pristupljeno: 14-lip-2018].
[20] F. Valsorda, „Escaping a chroot jail/1“, *PyTux*. [Na internetu]. Dostupno na: https://filippo.io/escaping-a-chroot-jail-slash-1/. [Pristupljeno: 14-lip-2018].
[21] „Breaking out of a chroot() jail“, 12-svi-2002. [Na internetu]. Dostupno na: https://web.archive.org/web/20131203024850/http://www.bpfh.net/computing/docs/chroot-break.html. [Pristupljeno: 14-lip-2018].
[22] P.-H. Kamp i R. N. M. Watson, „Jails: Confining the omnipotent root.“, str. 15.
[23] M. Riondato, „Chapter 14. Jails“, *FreeBSD Handbook*. [Na internetu]. Dostupno na: https://www.freebsd.org/doc/en_US.ISO8859-1/books/handbook/jails.html. [Pristupljeno: 14-lip-2018].
[24] J. Beck *i ostali*, „Virtualization and Namespace Isolation in the Solaris Operating System“. 07-ruj-2006.
[25] J. Frazelle, „Setting the Record Straight: containers vs. Zones vs. Jails vs. VMs“, *Jessie Frazelle's Blog*, 28-ožu-2017. [Na internetu]. Dostupno na: https://blog.jessfraz.com/post/containers-zones-jails-vms/. [Pristupljeno: 14-lip-2018].
[26] „docker - Difference between LXC and libcontainer“, *Stack Overflow*, 08-pros-2015. [Na internetu]. Dostupno na: https://stackoverflow.com/questions/34152365/difference-between-lxc-and-libcontainer/34155329. [Pristupljeno: 16-lip-2018].
[27] „Docker frequently asked questions (FAQ)“, *Docker Documentation*, 16-lip-2018. [Na internetu]. Dostupno na: https://docs.docker.com/engine/faq/. [Pristupljeno: 16-lip-2018].
[28] T. Taylor, „Docker vs. LXC—The Similarities and Differences“, *wercker*, 14-ožu-2018. [Na internetu]. Dostupno na: http://blog.wercker.com/what-is-the-difference-between-lxc-and-docker. [Pristupljeno: 16-lip-2018].
[29] „Docker vs LXC....What is the consensus on this? • r/linuxadmin“, *reddit*, 22-sij-2018. [Na internetu]. Dostupno na: https://www.reddit.com/r/linuxadmin/comments/7s9ctv/docker_vs_lxcwhat_is_the_consensus_on_this/. [Pristupljeno: 16-lip-2018].
[30] „OpenVZ Virtuozzo Containers Wiki“. [Na internetu]. Dostupno na: https://openvz.org/Main_Page. [Pristupljeno: 16-lip-2018].
[31] „rkt“, *CoreOS*. [Na internetu]. Dostupno na: https://coreos.com/rkt/. [Pristupljeno: 16-lip-2018].
[32] M. Long, „What Is Container Linux and Should You Use It?“, *MakeUseOf*, 24-ožu-2017. [Na internetu]. Dostupno na: https://www.makeuseof.com/tag/what-is-container-linux/. [Pristupljeno: 16-lip-2018].
[33] B. Hemphill, „What is the significance of CoreOS?“, *Quora*, 16-lip-2015. [Na internetu]. Dostupno na: https://www.quora.com/What-is-the-significance-of-CoreOS. [Pristupljeno: 16-lip-2018].

[34] A. Polvi, „CoreOS to join Red Hat to deliver automated operations to all“, *CoreOS*, 30-sij-2018. [Na internetu]. Dostupno na: https://coreos.com/blog/coreos-agrees-to-join-red-hat?utm_source=blog&utm_medium=referral. [Pristupljeno: 16-lip-2018].
[35] „Production-Grade Container Orchestration“, *Kubernetes*. [Na internetu]. Dostupno na: https://kubernetes.io/. [Pristupljeno: 16-lip-2018].
[36] „What is Kubernetes?“, *Kubernetes*. [Na internetu]. Dostupno na: https://kubernetes.io/docs/concepts/overview/what-is-kubernetes/. [Pristupljeno: 16-lip-2018].
[37] A. Yigal, „The Rise of Kubernetes Popularity in 2017“, *Logz.io*, 26-velj-2018. [Na internetu]. Dostupno na: https://logz.io/blog/rise-kubernetes-2017/. [Pristupljeno: 16-lip-2018].
[38] S. Yegulalp, „What is Kubernetes? Container orchestration explained“, *InfoWorld*, 04-tra-2018. [Na internetu]. Dostupno na: https://www.infoworld.com/article/3268073/containers/what-is-kubernetes-container-orchestration-explained.html. [Pristupljeno: 27-lip-2018].
[39] L. Liu i M. Brown, „Containerd Brings More Container Runtime Options for Kubernetes“, *Kubernetes*, 02-stu-2017. [Na internetu]. Dostupno na: https://kubernetes.io/blog/2017/11/containerd-container-runtime-options-kubernetes/. [Pristupljeno: 16-lip-2018].
[40] S. Simeonova, „Threat Modeling in the Enterprise, Part 1: Understanding the Basics“, *Security Intelligence*, 08-kol-2016. [Na internetu]. Dostupno na: https://securityintelligence.com/threat-modeling-in-the-enterprise-part-1-understanding-the-basics/. [Pristupljeno: 12-lip-2018].
[41] „Threat model“, *EFF Surveillance Self-Defense*. [Na internetu]. Dostupno na: https://ssd.eff.org/en/glossary/threat-model. [Pristupljeno: 12-lip-2018].
[42] „Application Threat Modeling“, *OWASP*. [Na internetu]. Dostupno na: https://www.owasp.org/index.php?title=Application_Threat_Modeling&oldid=241151. [Pristupljeno: 12-lip-2018].
[43] J. Steven, „What is threat modeling? A vocabulary of threat model terms“, *Synopsys*, 11-svi-2011. [Na internetu]. Dostupno na: https://www.synopsys.com/blogs/software-security/threat-modeling-vocabulary/. [Pristupljeno: 12-lip-2018].
[44] C. Salter, J. Wallner, B. Schneier, i O. S. Saydjari, „Toward A Secure System Engineering Methodology“, predstavljeno na New Security Paradigms Workshop, 1998.
[45] J. Schaumann, „Know Your Enemy - An Introduction to Threat Modeling“, *Signs of Triviality*, 05-pros-2016. [Na internetu]. Dostupno na: https://www.netmeister.org/blog/threat-model-101.html. [Pristupljeno: 12-lip-2018].
[46] J. D. Meier, A. Murukan, R. Escamilla, S. Vasireddy, M. Dunner, i A. Mackman, „Threat Modeling“, *Improving Web Application Security: Threats and Countermeasures*, lip-2003. [Na internetu]. Dostupno na: https://docs.microsoft.com/en-us/previous-versions/msp-n-p/ff648644(v%3dpandp.10). [Pristupljeno: 12-lip-2018].
[47] „Revision history of ‚Application Threat Modeling‘ - OWASP“. [Na internetu]. Dostupno na: https://www.owasp.org/index.php?title=Application_Threat_Modeling&action=history. [Pristupljeno: 13-lip-2018].
[48] A. Shostack, *Threat Modeling: Designing for Security*. Indianapolis, IN: Wiley, 2014.
[49] „Threat Risk Modeling“, *OWASP*. [Na internetu]. Dostupno na:

https://www.owasp.org/index.php?title=Threat_Risk_Modeling&oldid=231638. [Pristupljeno: 12-lip-2018].
[50] Rodrigo Santos, „Threats - Microsoft Threat Modeling Tool - Azure“. [Na internetu]. Dostupno na: https://docs.microsoft.com/en-us/azure/security/azure-security-threat-modeling-tool-threats. [Pristupljeno: 12-lip-2018].
[51] A. Shostack, „Threat Modeling: What, Why, and How?“, *MISTI Training Institute*, 21-pros-2017. [Na internetu]. Dostupno na: https://misti.com/infosec-insider/threat-modeling-what-why-and-how. [Pristupljeno: 12-lip-2018].
[52] B. Beyst, „Which Threat Modeling Methodology | STRIDE, VAST, PASTA, Trike“, *ThreatModeler Software, Inc.*, 15-tra-2016. [Na internetu]. Dostupno na: https://threatmodeler.com/2016/04/15/threat-modeling-methodology/. [Pristupljeno: 12-lip-2018].
[53] J. Stanganelli, „Which Threat Risk Model Is Right for Your Organization?“, *eSecurity Planet*, 19-ruj-2016. [Na internetu]. Dostupno na: https://www.esecurityplanet.com/network-security/which-threat-risk-model-is-right-for-your-organization.html. [Pristupljeno: 12-lip-2018].
[54] David LeBlanc, „DREADful“, *David LeBlanc's Web Log*, 14-kol-2007. [Na internetu]. Dostupno na: https://blogs.msdn.microsoft.com/david_leblanc/2007/08/14/dreadful/. [Pristupljeno: 12-lip-2018].
[55] „Threat model“, *Wikipedia*. 28-svi-2018.
[56] M. M. Morana, „Application Threat Modeling“, str. 41, 2015.
[57] „Trike: Trike“. [Na internetu]. Dostupno na: http://www.octotrike.org/. [Pristupljeno: 12-lip-2018].
[58] „Docker security“, *Docker Documentation*, 22-lip-2018. [Na internetu]. Dostupno na: https://docs.docker.com/engine/security/security/. [Pristupljeno: 22-lip-2018].
[59] „Understand container communication“, *Docker Documentation*, 04-svi-2018. [Na internetu]. Dostupno na: https://docs.docker.com/engine/userguide/networking/default_network/container-communication/. [Pristupljeno: 23-lip-2018].
[60] „Docker Hub“. [Na internetu]. Dostupno na: https://hub.docker.com/. [Pristupljeno: 27-lip-2018].
[61] „dockerd“, *Docker Documentation*, 22-lip-2018. [Na internetu]. Dostupno na: https://docs.docker.com/engine/reference/commandline/dockerd/. [Pristupljeno: 23-lip-2018].
[62] „NikolaKupec/docker-bench-security-python“, *GitHub*. [Na internetu]. Dostupno na: https://github.com/NikolaKupec/docker-bench-security-python. [Pristupljeno: 27-lip-2018].
[63] „CIS Docker Community Edition Benchmark v1.1.0“. Center for Internet Security, 06-srp-2017.
[64] A. Kili, „Learn Linux System Auditing with Auditd Tool on CentOS/RHEL“, *TecMint*, 22-ruj-2017. [Na internetu]. Dostupno na: https://www.tecmint.com/linux-system-auditing-with-auditd-tool-on-centos-rhel/. [Pristupljeno: 17-lip-2018].
[65] „Pipfile: the replacement for requirements.txt“, *GitHub*. [Na internetu]. Dostupno na: https://github.com/pypa/pipfile. [Pristupljeno: 18-lip-2018].
[66] G. Rodola', *psutil: Cross-platform lib for process and system monitoring in Python*. 2018.
[67] „Što je IEEE? - IEEE“, *Hrvatska sekcija IEEE-a*. [Na internetu]. Dostupno na:

http://www.ieee.hr/ieeesection/upoznajte__ieee/sto_je_ieee. [Pristupljeno: 24-lip-2018].
[68] „Chapter 1. Introduction to Control Groups (Cgroups)“, *Red Hat Customer Portal*. [Na internetu]. Dostupno na: https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/6/html/resource_management_guide/ch01. [Pristupljeno: 26-lip-2018].
[69] „namespaces(7)“, *Linux manual*. [Na internetu]. Dostupno na: http://man7.org/linux/man-pages/man7/namespaces.7.html. [Pristupljeno: 26-lip-2018].
[70] „Host name definition“, *The Linux Information Project*. [Na internetu]. Dostupno na: http://www.linfo.org/host_name.html. [Pristupljeno: 26-lip-2018].
[71] „Operacijski sustavi“, *FERWeb*. [Na internetu]. Dostupno na: https://www.fer.unizg.hr/predmet/os. [Pristupljeno: 24-lip-2018].

A. Rječnik

Administratorska prava (engl. *root privileges*) – prava potrebna korisniku računalnog sustava kako bi mogao izvršavati administrativne izmjene i postavke na računalnom sustavu – korisnik sa administratorskim pravima nije ni na koji način ograničen u svojoj uporabi računalnog sustava te se može koristiti svim računalnim resursima i raditi bilo kakve izmjene u računalnom sustavu

Biblioteke (engl. *libraries*) – kôd koji je osmišljen i zapakiran da bude na raspolaganju drugim programima – najčešće pruža neke tražene funkcionalnosti koje autori novih programa mogu iskoristiti

BSD – OS slobodnog kôda s izravnim porijeklom od starijih Unix OS-ova

Centar za internet sigurnost (engl. *Center for Internet Security*, *CIS*) – neprofitna organizacija posvećena zaštiti od računalnih prijetnji

Često postavljena pitanja (engl. *frequently asked questions*, *FAQ*) – popis često postavljenih pitanja u svezi s određenom temom i odgovori na ta pitanja

Chroot zatvor (engl. *chroot jail*) – virtualno ograđeno okružje za relativno blagu izolaciju procesa u OS-u – ograničava proces na podstablo domaćinovog datotečnog stabla

Datoteke uređaja (engl. *device files*) – datoteke u datotečnom sustavu koje predstavljaju sučelja prema uređajima sklopovlja ili sposobnostima OS-a

Dekorator funkcija (engl. *function decorator*) – funkcija u Pythonu koja kao parametar prima funkciju i vraća funkciju i koja se upotrebljava kod definicije neke druge funkcije sa posebnom sintaksom, koristeći znak @

Dijagram protoka podataka (engl. *data flow diagram*, *DFD*) – grafički prikaz protoka podataka kroz sustav

Dobro (engl. *asset*) – opipljiva ili neopipljiva vrijednost u sustavu koja može biti izložena prijetnjama i o čijoj sigurnosti valja razmišljati

Dokumentacijski tekst (engl. *docstring*) – komad teksta u Pythonu koji nije pridružen ni jednoj varijabli s namjerom da se koristi kao opis kôda koji ga okružuje – drugi programi ga mogu dohvatiti i od njega izraditi potpunu dokumentaciju kôda

Domaćin (engl. *host*) – računalo koje sadrži jedno ili više virtualiziranih ili kontejneriziranih programa ili sustava

Generatorska funkcija (engl. generator function) – funkcija u Pythonu koja umjesto jedne povratne vrijednosti može generirati niz povratnih vrijednosti koje je moguće dohvaćati jednu po jednu

Institut inženjera elektrotehnike i elektronike (engl. *Institute of Electrical and Electronics Engineers*, *IEEE*) – neprofitna organizacija i vodeći autoritet na širokom tehničkom području od računalnih znanosti, biomedicinske tehnike i telekomunikacija, preko električne energije, potrošačke elektronike te mnogih drugih područja [67]

Ishodišni direktorij (engl. *root directory*) – direktorij koji predstavlja ishodište jednog datotečnog sustava – svi ostali direktoriji i datoteke u datotečnom sustavu potomci su ishodišnog direktorija

Iznimka (engl. *exception*) – mehanizam u Pythonu koji omogućuje prekidanje uobičajenog toka kôda i prenošenje podataka o prekidu komadu kôda koji ga izričito očekuje – najčešće korišteno za vođenje brige o greškama koje se mogu dogoditi tokom izvršavanja

Izolacija programske potpore (engl. *software isolation*) – osiguravanje da računalni programi budu odvojeni jedni od drugih i da ne utječu jedni na druge

Kontejnerizacija (engl. *containerization*) – Relativno novi termin koji označava trend izolacije programske potpore unutar jednog OS-a – izolirana okruženja koja se izrađuju kontejnerizacijom nazivaju se kontejneri (engl. *container*)

Kontrolne grupe (engl. *control groups*, *cgroups*) – funkcionalnost jezgre Linux kojom se omogućuje postavljanje ograničenja procesima na korištenje resursa sustava [68]

Linux – vodeći OS slobodnog kôda izrađen po uzoru na OS-ove Unix tradicije

Linux kontejneri (engl. *Linux containers*, *LXC*) – implementacija izoliranja programske potpore, pod pokroviteljstvom tvrtke Canonical – nekoć su bili temelj Docker kontejnera

Među-kontejnerska komunikacija (engl. *inter-container communication*, *ICC*) – Dockerov naziv za omogućenu mrežnu komunikaciju između kontejnera na istom poslužitelju

Međukôd (engl. *byte-code*) – kôd koji predstavlja međukorak između programskih jezika više razine i strojnog kôda koji računalno sklopovlje može izvršavati

Mikroservisi (engl. *microservices*) – vrsta programske potpore oblikovana tako da se usredotočuje na vrlo usko područje čitavog problema i da svoju usko zadanu zadaću odradi temeljito – čitav problem se rješava korištenjem više takvih mikroservisa umjesto jedne monolitne aplikacije

Mrežni preglednik (engl. *web browser*) – računalni program koji omogućuje lakše pretraživanje i pregledavanje mrežnih stranica - najčešće dolazi sa naprednim grafičkim sučeljem

Mrežni usmjernici (engl. *routers*) – mrežni uređaji koji usmjeruju podatkovne pakete između računalnih mreža

Prostori imena (engl. *namespaces*) – funkcionalnost jezgre Linux kojom se omogućuje postavljanje procesa u prostore iz kojih imaju zaseban i ograničen pogled na resurse sustava [69]

Nakupina (engl. *cluster*) – skup povezanih računala koja zajedno funkcioniraju kao jedinica

Naziv domaćina (engl. *hostname*) – ime računala koje je spojeno na mrežu kojim ga je moguće jednoznačno prepoznati [70]

Operacijski sustav (engl. *Operating System*, *OS*) – skup programa koji djeluju kao posrednici između sklopovlja i primjenskih programa te korisnika [71]

Orkestriranje kontejnera (engl. container *orchestration*) – automatizacija stavljanja u pogon, upravljanja, skaliranja, umrežavanja i dostupnosti aplikacija građenih na kontejnerima [38]

OWASP – neprofitna organizacija koja se bavi informiranjem o sigurnosti računalnih aplikacija

POSIX (engl. *Portable Operating System Interface*) – skupina računalnih standarda izrađenih u IEEE-u s namjerom održavanjem kompatibilnosti između različitih OS-ova – najviše se odnose na Unix OS-ove

Postaviti (engl. *mount*) – odnosi se na postavljanje neke vrste pohrane podataka na raspolaganje korisnicima računalnog sustava

Pozivi jezgri OS-a (engl. *system call*) – zahtjeve koje šalje računalni program jezgri operacijskog sustava kada mu zatrebaju računalni resursi poput pohrane podataka ili prikazivanje slike na zaslonu ekrana – programi ne trebaju pozive jezgri za vrlo osnovne naredbe poput aritmetičkih operacija

Predaja (engl. commit) – spremanje komada programskog kôda u sustav za verzioniranje, obično popraćen kratkom porukom

Prenosivost (engl. *portability*) – sposobnost nekog programa ili komada kôda da bude iskorišten na različitim operacijskim sustavima i / ili sklopovljima

Programski jezik Python (engl. *Python programming language*) – dinamički, interpretirani programski jezik visoke razine

Programsko sučelje (engl. *application programming interface*, *API*) – skup jasno određenih metoda ili funkcija kojima različiti dijelovi programske potpore mogu međusobno komunicirati

Protokol TLS (engl. *TLS protocol*) – kriptografski protokol koji omogućuje sigurnu komunikaciju između dvije krajnje točke preko nezaštićene mreže

Python paket (engl. Python package) – skup Python kôda unutar jednog direktorija u kojem postoji Python modul po nazivu *__init__.py*.

Root korisnik ili **administratorski korisnik** (engl. *root user*) – korisnik sa administratorskim pravima

Seccomp – funkcionalnost jezgre Linux koja omogućuje ograničavanje koje pozive jezgri proces može izvršavati

Servis Docker (engl. *Docker daemon*) – program koji radi u pozadini i brine se o stanju, pokretanju i zaustavljanju Docker kontejnera – preko njega se upravlja sa svim aspektima Docker kontejnera

Solaris zone (engl. *Solaris Zones*) – implementacija izoliranja programa na operacijskom sustavu Solaris – nadahnuta FreeBSD zatvorima

Središnja jedinica za obradu (engl. *Central Processing Unit, CPU*) – često zvana samo *procesor* – središnji je dio računala odgovoran za izvršavanje osnovnih naredbi od kojih se računalni programi sastoje

Sustav za verzioniranje kôda (engl. *version control system*, *VCS*) – sustav koji omogućuje organizirano spremanje i upravljanje digitalnim podacima, najčešće programskim kôdom

Unix – stariji OS koji je nadahnuo novu porodicu operacijskih sustava – najpoznatiji OS-ovi koji proizlaze iz tradicije koju je Unix započeo su Linux, BSD i Appleov MacOS

Unix utičnice (engl. *UNIX socket*) – način na koji procesi na istom računalu mogu međusobno komunicirati

Vatrozidovi (engl. *firewalls*) – sustav koji upravlja i ograničava mrežni promet koji prolazi njime

Virtualizacija sklopovlja (engl. *hardware virtualization*) – emulacija fizičkog sklopovlja na kojoj je moguće izvršavati virtualne strojeve – ta se emulacija odvija na fizičkom računalu *domaćinu*, koji može sadržavati više virtualnih sklopovlja i virtualnih strojeva

Virtualni strojevi (engl. *virtual machines*) – zasebni računalni sustavi koji postoje na emuliranom (*virtualnom*) sklopovlju

Vjerodajnica (engl. *certificate*) – elektronički kriptografski dokument koji služi kao dokaz identiteta

Zaštićena okolina (engl. *sandbox*) – okolina za računalne procese koja ih izolira od ostatka sustava i omogućuje izvršavanje i istraživanje tih procesa bez izravne prijetnje ostatku sustava

Zavisnosti (engl. *dependencies*) – dijelovi kôda ili programska potpora koji su nužni određenom programu za normalno funkcioniranje – taj program je *zavisan* o tim dijelovima kôda ili programskoj potpori