VILNIAUS GEDIMINO TECHNIKOS UNIVERSITETAS

FUNDAMENTINIŲ MOKSLŲ FAKULTETAS

INFORMACINIŲ SISTEMŲ KATEDRA

VIRTUALIOS INFRASTRUKTŪROS IR DEBESŲ KOMPIUTERIJOS SAUGA

Docker platforma ir jos saugos aspektai

Darbą atliko: Vladislav Zmitrovič

Vilnius 2018

Turinys

1. Įvadas

Docker, tai programinė įranga skirta automatizuotam ir greitam programinių produktų virtualizavimui operacinės sistemos lygyje. *Docker* pagalba galima supakuoti programą su visa jos aplinką ir priklausomybėmis į konteinerį, kuris gali būti lengvai perkeltas į bet kuria *Linux* sistemą (nuo 1.6 versijos palaikoma ir *Windows* OS), kurios branduolys palaiko *cgroups* ir pateikia aplinką konteinerių valdymui.

Linux konteineris, tai procesų rinkinis, kuris yra atskirtas nuo likusios operacinės sistemos ir paleidžiamas iš atskiro atvaizdo, kuriame yra visi failai reikalingų šių procesų veikimui. Kadangi atvaizde yra saugomos visos programai reikalingos bibliotekos, atvaizdą galima lengvai perkėlinėti tarp skirtingų kompiuterių ir aplinkų (programavimo, testavimo, produkcinės).

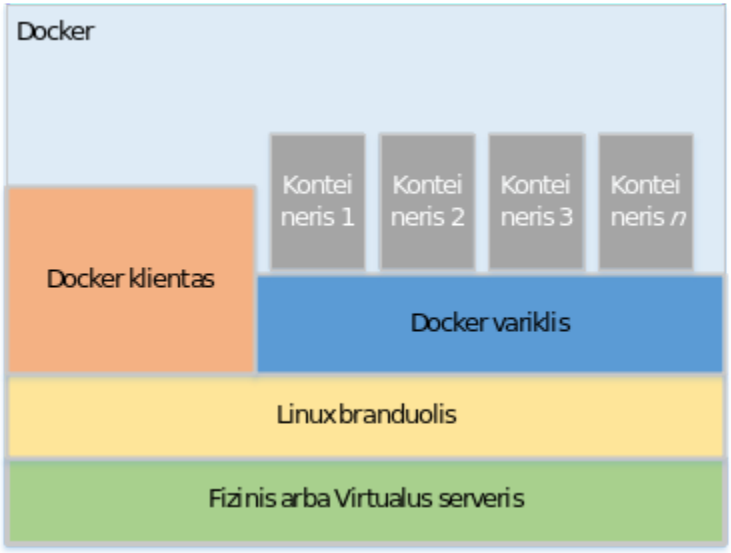
Konteinernizacija ir komponentų izoliavimas nėra naujovė IT pasaulyje, kai kurios *Unix* sistemos naudoja konteinerius jau daugiau kaip 10 metų. Pavyzdžiui *FreeBSD Jail* virtualizacijos mechanizmas.

Linux Containers sistema (LXC), tai konternizacijos technologijų pagrindas. Į *Linux* branduolį jį buvo įtraukta 2008 metais. Pagrindinis ir greitas žingsnis į izoliaciją buvo esamų technologijų integravimas. Visų pirma, *cgroups* mechanizmas veikiantis Linux branduolyje, ir apribojantis sistemos išteklių naudojimo procesą ar grupę procesų kartu su iniciacijos sistema *systemd*, atsakinga už vartotojo erdves sukūrimą ir procesų paleidimą. Šių mechanizmų derinys, kuris iš pradžių buvo sukurtas siekiant pagerinti bendrą *Linux* valdymą, leido žymiai geriau kontroliuoti atskirus procesus ir sukūrė pagrindą sėkmingam aplinkų atskyrimui.

Kitas įvykis konteinerių istorijoje yra susijęs su vartotojo vardų sričių vystymų, leidžiantis suskirstyti priskirtus procesams vartotojų identifikatorius į grupės viduje ir už vardų srities. Konteinerių kontekste tai reiškia, kad vartotojai ir grupės gali turėti teisės atlikti tam tikras operacijas konteinerio viduje, bet ne už jo ribų. Tai kažkiek panašu į *Jail* koncepciją, bet saugesnė dėl papildomos procesų izoliacijos.

Docker atsiradęs kiek vėliau buvo pozicionuojamas kaip įrankis skirtas palengvinti ir supaprastinti darbus susijusius su konteinerių kūrimu ir valdymu. Pradžioje *Docker* naudojo LXC kaip valdymo tvarkyklė pagal nutylėjimą. *Docker* praktiškai neatnešęs daug naujų idėjų bet supaprastinęs konteinerių valdymą ir standartizavus sąsają padarė konteinerius

prieinamus programuotojams ir administratoriams. Tai stimuliavo susidomėjimą konteinerių technologijomis tarp *Linux* programuotojų.



1.1.1.A.1 pav. *Docker struktūros schema*

2. Konteinerių nauda

Konteineriai turi savyje daug patrauklių pranašumu tiek programuotojams, tiek sistemų administratoriams. Kai kurie svariausi pranašumai aprašomi žemiau.

2.1. Sistemų abstrahavimas nuo konteinerizuotu programų

Konteineriai sukurti būti pilnai standartizuoti. Tai reiškia, kad konteineriai jungiasi prie sistemos arba kitų išorinių (konteinerio atžvilgiu) resursų standartizuotu sąsajų pagalba. Programa esanti konteineryje nepriklauso nuo resursų arba sistemos, kurioje yra konteineriai, konfigūracijos. Tai supaprastina programos kūrimo procesą. Analogiškai iš operacinės sistemos pusės, kurioje yra patalpinti *Docker* konteineriai, visi konteineriai atrodo vienodai ir sistemos nereikia papildomos konfigūracijos talpinant joje skirtingus konteinerius.

2.2. Plečiamumo paprastumas

Vienas iš abstrahavimo tarp operacinės sistemos ir konteinerių privalumu yra tas, kad pritaikius tinkamą dizainą, mastelio keitimas gali būti ganėtinai paprastas. Į paslaugas orientuota architektūra kartu su konteineriais pritaikytomis programomis suteikia pagrindą patogiam sistemų plečiamumui.

Programuotojas arba administratorius gali paleisti kelius konteinerius savo darbiniame kompiuteryje, tuo pačiu tokia pati sistema gali būti patalpinta ir horizontaliai išplėsta pvz. testavimo aplinkoje. Perkeliant konteinerius į produkciją jie gali būti dar labiau išplėsti neatliekant jokių pakeitimų pačiose konteineriuose arba juose patalpintuose aplikacijose.

2.3. Lengvas priklausomybių ir programų versijų valdymas

Konteineriai leidžia programuotojui susieti programą arba programos komponentą su visomis jo priklausomybėmis ir toliau dirbti su jais kaip vieną visumą. Operacinei sistemai nereikia nerimauti dėl priklausomybių, reikalingų korektiškam programos veikimui. Jei operacinė sistema gali paleisti *Docker* konteinerį, ji gali paleisti bet kokį *Docker* konteinerį.

Tai leidžia lengvai valdyti priklausomybes ir supaprastina programos versijų kontrolę. Operacinės sistemos daugiau nėra atsakingos už programų priklausomybių valdymą, nes, išskyrus tuos atvejus, kai vieni konteineriai priklauso nuo kitų konteinerių, visos priklausomybės turi būti pačiame konteineryje.

2.4. Izoliuotos ir „lengvos“ vykdymo aplinkos

Nepaisant to, kad konteineriai nesuteikia tokio pat izoliacijos ir resursų valdymo lygio kaip virtualizavimo technologijos, jie turi labai „lengva“ (kompiuterinių resursų atžvilgiu) vykdomąja aplinką. Konteineriai yra izoliuojami procesų lygyje, dirbdami to paties pagrindinio branduolio paviršiuje. Tai reiškia, kad konteineryje nėra pilnavertes operacinės sistemos, kuo pasėkoje konteineris paleidžiamas labai greitai. Programuotojai lengvai gali paleisti šimtus konteinerių savo darbo kompiuteryje be jokių problemų.

2.5. Bendrai naudojami sluoksniai

Konteineriai yra „lengvi“ taip pat dėl to, kad jie yra išsaugomi „pasluoksniui“. Jeigu keli konteineriai yra sukurti tuo pačio sluoksnio pagrindu, jie gali bendrai naudotis tuo sluoksniu be jo kopijavimo. Kuo pasėkoje sumažėja disko vietos panaudojimas naujuose konteinerių atvaizduose.

2.6. Komponavimo galimybės ir nuspėjamumas

Docker failai leidžia vartotojams nurodyti konkrečius veiksmus, kurios reikia atlikti norint sukurti naują konteinerio atvaizdą. Tai leidžia nustatyti vykdymo aplinkos nustatymus taip, tarsi jis būtų programos kodas, tuo pačiu išsaugant šiuos parametrus versijų valdymo

sistemoje. Tas pats *Docker* failas, surinktas toje pačioje aplinkoje, visada sukuria identišką konteinerio atvaizdą.

3. Docker programų architektūra

Kuriant programas, kurios bus naudojamos Docker konteineriuose, vienas iš pirmųjų klausimų, tai programos architektūra. Paprastai konteineriams skirtos programos geriausiai dirba su į paslaugas orientuota architektūrą.

Į paslaugos orientuotos programos padalina sistemos funkcionalumą į atskirus komponentus, kurie bendrauja tarpusavyje per aiškiai apibrėžtas sąsajas. Konteinerių technologija pati skatina tokio tipo dizainą, nes tai leidžia pačiam išplėsti arba atnaujinti kiekvieną komponentą.

Programos įgyvendinančios tokį požiūrį į architektūrą turi turėti šias charakteristikas:

- Jie neturėtu pasikliauti sistemos ypatumais.
- Kiekvienas komponentas turi pateikti nuoseklų API, kurį naudotojai gali naudoti norėdami naudotis paslauga.
- Kiekvienas servisas pradinės konfigūracijos metu turi atsižvelgti į aplinkos kintamuosius.
- Programos turi būti saugomos už konteinerio ribų – sumontuotose saugyklose, arba atskiruose konteineriuose su duomenimis.

Šių taisyklių laikymasis leidžia keisti kiekvieną komponentą nepriklausomai (bent jau tol, kol yra palaikomas API). Jos taip pat fokusuojasi į horizontalų plečiamumą, nes kiekvienas komponentas gali būti praplėstas aptikus siauras vietas.

Konfigūracijos iškėlimas už konteinerio ribų suteikia galimybę keisti konfigūraciją nekeičiant konteinerio atvaizdo. Taip pat tai leidžia redaguoti keliu konteineriu konfigūraciją centralizuotai.

4. Saugumas

Konteinerių technologija keičia nusistovėjusį požiūrį į saugumą. Naujo požiūrio esme tame, kad turint atvaizdą, kuriame yra tik būtini komponentai ir kuris paleidžiamas tik tada, kai to reikia. Nėra jokios papildomos programinės įrangos, kuri gali turėti saugumo spragų. Iš saugumo pusės, kuo mažiau yra paleista procesų tuo didesnis saugumas. Tačiau netgi jeigu

konteineryje veikia tik kelios programos, vis tiek reikia įsitikinti, kad jos yra laikų atnaujinamos ir neturi viešai paskelbtų pažeidžiamumų.

Kai *Docker* pirmą kartą buvo paleistas 2013 m., jam trūko patikimų saugos funkcijų ir įrankių, kurie leistų naudoti konteinerius įmonėse ir atitiktų verslo keliamus saugumo reikalavimus.

Nuo 2013 metų padėtis pasikeitė. Iš vienos pusės *Docker* bendruomene deda nemažai pastangų saugumo padidinimui. Iš kitos pusės, priklausomai nuo to, kaip planuojama naudoti konteinerius yra papildomos priemonės skirtos padidinti *Docker* saugumą.

Pradėjus augti *Docker* populiarumui atsirado įrankiai skirti konteinerių saugumui.

Vieni iš svarbesnių įrankių, tai atvaizdų skeneriai, pvz. *Clair, Docker Security Scanning*, kurie yra integruoti į *Docker Hub* ir gali automatiškai aptikti atvaizduose esamus žinomus pažeidžiamumus. Tokie sprendimai padeda sutaupyti daug laiko. Jie taip pat gali siusti informaciją apie naujus pažeidžiamumus į el. paštą ir savarankiškai ieško saugumo atnaujinimų.

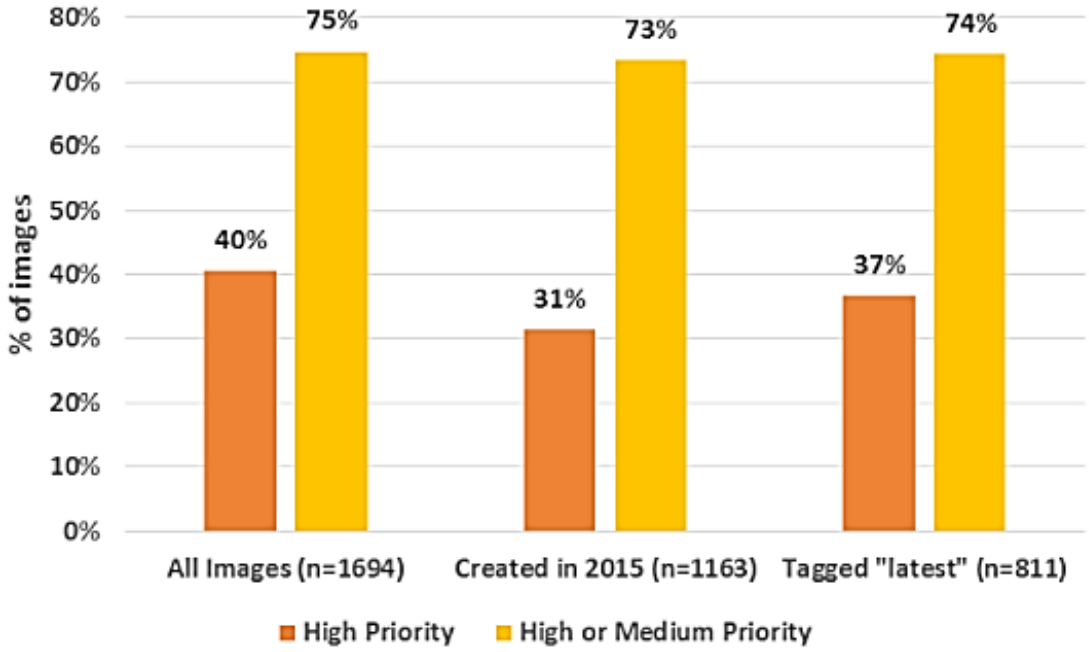
Yra dar vienas sprendimas skirtas konteinerių saugumui užtikrinti – *Twistlock*. Jis užtikrina visapusišką konteinerių apsaugą ir apima daugeli saugumo sričių. Jis ne tik apsaugo konteinerius, bet ir užtikrina analizė, stebėjimą ir savalaikišką reagavimą į atsiradusias grėsmės. *Tvistlock* yra vienas ir nedaugelio komercinių sprendimų, kurie orientuojasi išskirtinai į *Docker* saugumo užtikrinimą.

Per pastaruosius keletą metų *Docker* išleido eilė saugumo atnaujinimų, kas padėjo išspręsti dauguma saugumo spragų, dėl kurių įmonės nerizikavo naudoti *Docker* savo infrastruktūroje. Didesnis dėmėsis saugumui iš *Docker* pusės padidino įmonių susidomėjimą konteinerių technologijomis.

Pagrindinės saugumo problemos, kurios nekėlė pasitikėjimo buvo vardų sritys (*namespaces*) ir *cgroups*. Vardų sritys, tai paprasčiausias konteinerių izoliavimo būdas, užkertantis kelią jų sąveikai tarpusavyje. Vardų sritys leidžia naudoti skirtingas privilegijas, kurios gali būti priskirtos skirtingiems vartotojams. *Cgroups* riboja resursus, prieinamus kiekvienam konteineriui. Jie leidžia apriboti kiekvieno konteinerio prieigą prie darbinės atminties ir procesoriaus.

Docker bendruomene taip pat yra sukūrusi *Docker Bench,* scenarijų, gali patikrinti konteinerių ir kompiuterių konfigūracijos atitikimus geriausioms praktikoms pateiktomis *Center for Internet Security.*

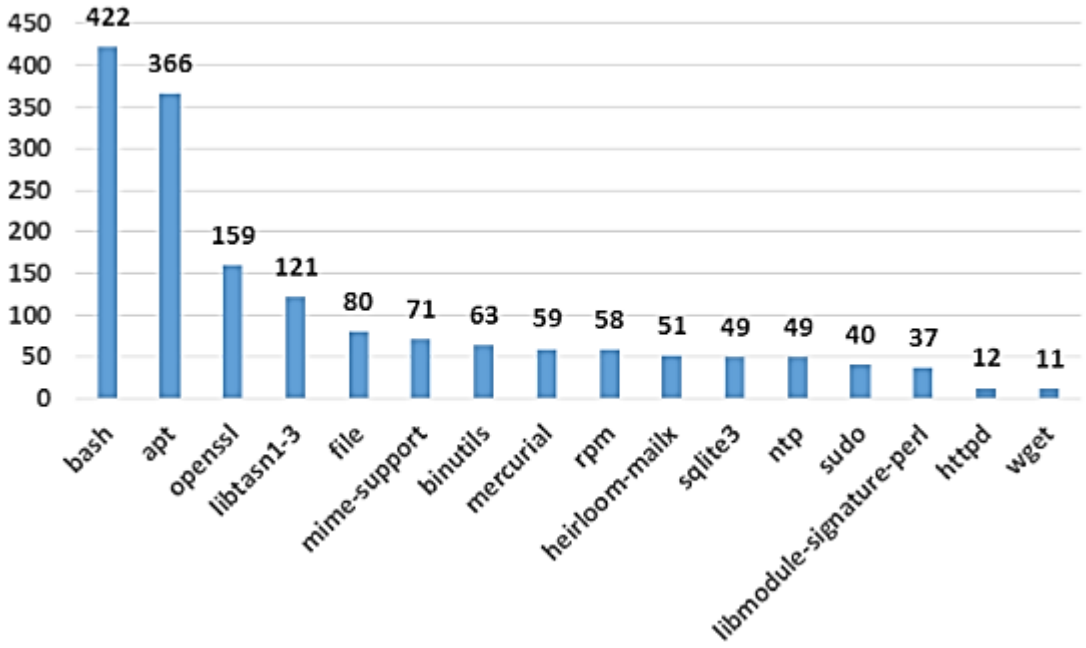
Bet nepaisant to saugumas *Docker* konteineriai dažnai turi pažeidžiamų elementų. Pagal *Banyan* ataskaitoje (pav. 4.1.) pateiktus duomenys už 2015 metus virš 40% oficialių atvaizdų iš *Docker Hub* turi pažeidžiamumų.



4.1.1.A.1 pav. *Docker atvaizdai su pažeidžiamumais*

Atliekant tyrimą buvo atsitiktinai pasirinkti ir ištirti maždaug 1700 atvaizdų. Beveik 40% atvaizdų turi aukšto lygio pažeidžiamumus. Kadangi į nėra jokių patikrinimų prieš įkeliant *Docker* atvaizdą į *Docker Hub* saugyklą, toks rezultatas nėra stebinantis.

Išanalizavus paketus, kurie turi įtaką daugeliui atvaizdų ir atsižvelgiant tik į tuos, kurie turi aukšto reitingo saugumo spragas, pastebėta, jog dažniausia sutinkamos bash, apt, openssl programos (pav. 4.2.).



4.1.1.A.2 pav. *Dažniausia pasitaikančios programos su žinomais pažeidžiamumais*

4.1. Docker konteinerių saugumo gerosios praktikos

Vienas iš pagrindinių virtualių mašinų ir konteinerių skirtumų yra pagrindinės sistemos branduolio naudojimas. Pagal numatytuosius nustatymus "Docker" konteineriai paleidžiami su aukščiausiomis privilegijomis, kurios gali sukelti rimtų problemų kadangi root vartotojo teisėmis dirbantis konteineris turi pilna prieigą prie sistemos, kurioje jis dirba.

4.1.2. Nenaudoti root vartotojo konteineriuose

Rizikas galima sumažinti paleidus konteinerį su paprasto vartotojo teisėmis. Štai kaip galima pakeisti vartotoją (pavyzdis su Rails programa):

```
# Sukuriamas darbinis katalogas
WORKDIR /app
# Kopijuojamas Rails-programos kodas į atvaizdą
COPY . ./
# Sukuriamas paprastas vartotojas ir grupė
RUN addgroup rails && adduser -D -G rails rails \
 && chown -R rails:rails /app
USER rails
```

4.1.3. Atlikinėti periodiška saugumo skenavimą

Taip pat svarbu atlikinėti saugumo skenavimą. Konteinerių rejestrai turi galimybė skenuoti įkeliamus atvaizdus. Pavyzdžiui *Docker* atlieka saugumo skenavimą visų oficialių ir vartotojų sukurtu *Docker* atvaizdų įkeliamu į *Docker Cloud*.

Atvaizdu saugumo skenavimui Quay.io naudoja *Clair* – atviro kodo projektą sukurta *CoreOS* kompanijos. Be *Clair* yra ir kitu saugumo skenerių, pvz. *TwistLock* ir *Aqua*, bet jie yra mokami.

Clair, tai programa parašyta *Golang* kalba, kuri realizuoja HTTP API rinkinį skirta atvaizdų įkėlimui, iškėlimui ir analizei. Pažeidžiamumų duomenys yra gaunami iš skirtingų šaltinių, tokių kaip *Debian Security Tracker* arba *RedHat Security Data* ir išsaugomi *Postgres* duomenų bazėje. *Clair* veikia statinės analizės principų, todėl tam, kad atlikti atvaizdo skenavimą jo nereikia paleisti, skenuojama tik atvaizdo failų sistema.

4.1.4. Naudoti Docker kartu su AppArmor/SELinux/TOMOYO

Ubuntu yra platinamas kartu su *AppArmor* šablonais. Visada rekomenduojama žinoti, ką veikia naudojama programinė įranga. Reikia tiksliai žinoti, kokios prieigų teisės yra reikalingos, prie kokiu direktorijų ir dokumentų programai reikalinga prieiga. Turint tokia informaciją galima apriboti prieigas iki reikiamo minimumo, taip užkertant kelią neteisėtiems veiksmams, kaip pvz. informacijos nutekėjimui arba teisių eskalavimui.

Norint atitikti keliamus reikalavimus būtina stebėti esamus konteinerius ir juose veikiančias programas nuo pat jų paleidimo. Tam tikslui kiekvieno konteinerio konfigūracijoje gali būti parašytas atitinkamas AppArmor profilis.

4.1.5. Naudokite seccomp syscals apribojimui

Seccomp palaikymas yra prieinamas *CentOS*, *Debian*, *Fedora*, *Gentoo*, *Oracle*, *Plamo* ir *Ubuntu* sistemose. Galima naudoti *seccomp* pakeičiant konteinerio konfigūraciją ir apibrėžiant naudojama *seccomp* taisyklė:

```
lxc.seccomp = /usr/share/lxc/config/common.seccomp
```

Docker galima pajungti šitą funkcionalumą aktyvuojant *–lxc-conf* parametrą.

4.1.6. Apriboti tinklo prieigas su iptables

Pagal nutylėjimą visi konteineriai naudoja *docker0* tinklo sąsają. Kaip ir bet kuriai kitai tinklo sąsajai *iptables* programa gali kontroliuoti tinklo prieigą konteineriams.

4.1.7. Nenaudoti SSH

Saugumui padidinti geriausia nenaudoti SSH konteineriuose. Vietoje jo geriau naudotis *docker exec -it mycontainer bash* komanda.

4.1.8. Visada naudoti naujausia Docker versiją

Dauguma programų turi klaidų ir saugumo spragų. *Docker* pastoviai yra tobulinamas. Klaidos yra taisomos ir pašalinamos žinomos saugumo spragos. Todėl yra labai svarbu laikų atnaujinti turimus konteinerius.

5. Išvados

Docker užtikrina pagrindą reikalinga paskirstytam konteinerių platinimui. Programos komponentų pakavimo į atskirus konteinerius pagalba, horizontalus plėtimasis tampa daug paprastesnis. Tam pakanka tik paleisti naujus konteinerius su reikiamais elementais. *Docker* teikia būtinus įrankius ne tik konteinerių kūrimui, bet ir jų valdymui bei dalijimuisi su kitais naudotojais ar kompiuteriais/serveriais.

Nors konteinerizuotos programos užtikrina reikiama procesų izoliaciją ir programų „pakavimą“, egzistuoja daugelis kitų komponentų reikalingų valdyti ir plėsti konteinerius paskirstytuose kompiuterių tinkluose.

Pagal savo architektūrą konteineriai turėtu užtikrinti pakankamą saugumo lygį. Bet dažnai konteineriuose būna patalpinti nereikalingi arba turintis žinomu pažeidžiamumų elementai. Tai yra viena iš silpniausiu konteinerių saugumo vietų. Kita – perteklinės prieigos teisės.

Tvarkingai sukonfigūravus ir laikų atnaujinant konteinerius bei sistemą, kurioje jie yra patalpinti galima užtikrinti reikiamą konteinerių saugumo lygį.

6. Literatūra

Gallagher, S. (2016). *Securing Docker.* Packt Publishing.

Schenker, G. N. (2018). *Learn Docker - Fundamentals of Docker 18.x.* Packt Publishing.

Turnbull, J. (2014). *The Docker Book.* Turnbull Press.