Pencegahan Serangan Permukaan Terhadap Docker Daemon Menggunakan Mode Rootless

Reyhan Rahmansyah[1], Vera Suryani[2], Fazmah Arif Yulianto[3]

[1,2,3]Fakultas Informatika, Universitas Telkom, Bandung
[1]reyhanrahmansyah@students.telkomuniversity.ac.id, [2]verasuryani@telkomuniversity.ac.id,
[3]fazmaharif@telkomuniversity.ac.id

Abstrak

Teknologi *containerization* menjadi salah satu alternatif dalam virtualisasi. Docker membutuhkan *docker daemon* untuk membangun, mendistribusikan, dan menjalankan *container* sehingga membuat docker tidak aman karena *docker daemon* rentan terserang oleh serangan permukaan (*Docker Daemon Attack Surface*). Serangan tersebut ialah serangan terhadap *docker daemon* yang mengambil alih akses (*root*). Pencegahan serangan dilakukan menggunakan *rootless mode*. Dalam tugas akhir ini, dilakukan pencegahan serangan *docker daemon attack surface* dengan membuat dan menjalankan *docker container* lalu mencegah serangan tersebut menggunakan *docker* dalam *rootless mode* sehingga seragan gagal dilakukan. Pembuktian bahwa serangan berhasil ialah pengguna dapat mengakses file /etc/shadow yang seharusnya file tersebut hanya dapat diakses oleh *user* yang mempunyai hak akses *root*. Didapatkan pernyataan bahwa file tersebut tidak dapat diakses jika docker dijalankan dengan *rootless mode*. Untuk mengukur apakah penggunaan *rootless mode* pada docker ini menambah beban CPU *usage* dan seberapa besar peningkatannya, maka dilakukan pengukuran CPU usage saat serangan dilakukan dengan docker yang dijalankan melalui hak akses root dan *rootless mode*. Didapatkan hasil penggunaan CPU sebesar 39% saat menggunakan docker dengan *rootless mode*. Sedangkan menggunakan docker dengan hak akses root hanya sebesar 0%. Peningkatan yang terjadi sebesar 39% merupakan peningkatan yang sepadan dengan keuntungannya yang dapat mencegah serangan *docker daemon attack surface*.

Kata kunci : docker, *container*, *daemon*, *rootless*, *root*, *privilege*.

Abstract

Containerization technology is an alternative in virtualization. Docker requires a docker daemon to build, distribute, and run containers so it makes the docker unsafe because the docker daemon is vulnerable to attack by the surface attacks (Docker Daemon Attack Surface). The attack is an attack on the docker daemon which takes over access (root). Prevention of attacks is done using by rootless mode. In this final project, docker daemon attack surface is prevented by creating and running a docker image and then preventing the attack using the docker in a rootless mode so the attack fails. Proof that the attack was successful is that the user can access the /etc/shadow file, which should only be accessible by users who have root privileges. Obtained a statement that the file cannot be accessed if the docker is running in rootless mode. To measure whether the use of the rootless mode on this docker adds to the burden of CPU usage and how much it increased, the CPU usage is measured when an attack is carried out with a docker that is run through root and rootless mode permissions. Obtained CPU usage results of 39% when using the docker with rootless mode. Whereas using a docker with root privileges is only 0%. The increase that occurred by 39% is an increase commensurate with its benefits that can prevent docker daemon attack surface.

Keywords: docker, *container*, *daemon*, *rootless*, *root*, *privilege*.

1. Pendahuluan

Latar Belakang

Pada beberapa tahun ini teknologi kontainerisasi telah semakin popular. Kepopularitasan ini dikarenakan teknologi ini memerlukan sumber daya yang sedikit, mudah diakses, dan rendah akan *error* atau *lag* pada saat digunakan. Banyak perusahaan yang menggunakan teknologi *container* ini, khususnya pada proses pengujian (*testing*) dan proses pengawakutuan (*debugging*). Contohnya adalah docker yang digunakan untuk mencoba aplikasi yang telah dibuat apakah dapat dijalankan di berbagai *platform* atau tidak dan jika ditemukan kerusakan (*bug*) dalam suatu program docker dapat digunakan dalam proses *debugging*. Namun teknologi *container* ini masih belum banyak digunakan pada proses produksi. Karena masih ada celah dalam bidang keamanannya.

Docker menggunakan *client* dan *server*. *Docker client* mengirimkan *request* ke *docker daemon* untuk membangun, mendistribusikan, dan menjalankan *docker container*. Pada proses mengirimkan *request* ke *docker daemon*, docker rentan terkena serangan *docker daemon attack surface* yang memungkinkan terambilnya hak akses *root* pada saat proses pengiriman *request* dijalankan. *Surface* yang dimaksudkan adalah hak akses *root* untuk

menggunakan docker. *User* yang memiliki akses ke *docker host* dan *docker daemon* secara otomatis mendapatkan kontrol penuh dari semua *container* dan *images* yang pada docker. *User* dengan hak akses root dapat membuat dan menghentikan *container*, menghapus *images*, memberikan perintah kepada *container* yang sedang berjalan, dan mengekspos informasi yang sensitif seperti *password* dan data-data lainnya. Untuk mencegah serangan itu terjadi, dilakukan pencegahan dengan menggunakan *rootless mode*. Dimana dalam mode tersebut tidak diperlukan akses *root* dalam penggunaan docker.

Rootless mode pada teknologi *container* berfungsi untuk menjalankan docker sebagai pengguna non-*root* pada *host* dan melindungi *host* dari potensi terjadinya serangan terhadap docker.

Topik dan Batasannya

Tugas akhir ini difokuskan pada proses pencegahan serangan *docker daemon attack surface* yang termasuk serangan dari *malicious insider* dan pencegahannya menggunakan *rootless mode* dan pengukuran CPU *usage* yang akan dibandingkan pada saat proses penggunaan docker berlangsung. Lalu berdasarkan data CPU *usage* yang telah diukur, akan didapatkan seberapa besar peningkatan CPU *usage* pada *rootless mode* dan apakah peningkatan tersebut sepadan dengan keuntungan yang didapatkan.

Untuk perumusan masalah yang akan dicangkup dalam penelitian ini adalah bagaimana mencegah serangan *docker daemon attack surface*, menunjukkan bahwa *rootless mode* pada docker lebih aman daripada menggunakan docker dengan hak akses *root*, dan pengujian performansi CPU *usage* menggunakan perintah 'docker stats' pada saat proses penggunaan docker dengan hak akses *root* dan docker dengan *rootless mode*. Diharapkan *rootless mode* dapat mencegah serangan *docker daemon attack surface*, sehingga dapat menghasilkan docker yang lebih aman.

Terdapat beberapa batasan yang ada pada tugas akhir ini, yaitu sistem ini hanya dapat digunakan untuk docker. *Rootless mode* hanya digunakan pada saat proses pencegahan serangan dan serangan *docker daemon attack surface* hanya terjadi pada docker dan menyerang *docker daemon* untuk memperoleh hak akses *root*. Jika serangan berhasil dilakukan, maka dibuktikan dengan *user* dapat mengakses file */etc/shadow* yang seharusnya hanya dapat diakses oleh *user* yang mempunyai hak akses *root*.

Tujuan

Tujuan dari tugas akhir ini adalah untuk mencegah serangan *docker daemon attack surface* pada docker dengan menggunakan *rootless mode* dan mengukur penambahan beban CPU *usage* pada *rootless mode*. Data CPU *usage* akan diambil pada saat docker dijalankan dengan hak akses *root* dan *rootless mode* dalam jangka waktu sesingkat-singkatnya agar didapatkan hasil yang akurat lalu dibandingkan apakah pada *rootless mode* terjadi peningkatan beban CPU *usage* dan apakah peningkatan tersebut sepadan dengan keuntungan yang didapatkan. Pengujian serangan dilakukan terhadap dua *docker container* yang dijalankan secara bergantian, yaitu *docker container* yang dijalankan dengan hak akses *root* dan *docker container* yang dijalankan dengan *rootless mode*.

Organisasi Tulisan

Bagian-bagian selanjutnya pada tugas akhir ini akan memaparkan mengenai dasar teori terkait sistem pencegahan serangan yang akan dibangun pada bagian 2. Kemudian dilanjut dengan pembahasan mengenai perancangan dan pembangunan sistem pencegahan serangan pada bagian 3. Pada bagian 4 akan ditunjukkan hasil pengujian dan evaluasi sistem. Lalu pada bab 5 akan membahas kesimpulan dan saran dari penelitian tugas akhir ini.

2. Dasar Teori

2.1 Docker

Docker merupakan suatu *platform* terbuka dalam bentuk teknologi virtualisasi berbasis *container*, ditujukan bagi para *developer* perangkat lunak dan pengelola sistem jaringan untuk membangun, mengirim, mem-*bundle* dan menjalankan aplikasi-aplikasi terdistribusi. Dalam bukunya yang berjudul "*Build Own PaaS with Docker*". Oskar Hane mengatakan bahwa docker adalah suatu cara untuk memasukkan layanan ke dalam lingkungan yang terisolasi yang disebut *container*, sehingga layanan tersebut dapat dikemas menjadi satu bersama dengan semua pustaka dan perangkat lunak lain yang dibutuhkan dan juga dapat dipastikan bahwa layanan akan berjalan dimanapun docker dijalankan (Hane, 2015).

Docker menggunakan fitur dari linux kernel seperti *namespace* dan *cgroups* (*control groups*) untuk membuat *container* diatas sebuah sistem operasi. *Namespaces* merupakan *building blocks* dari suatu *container*. Ada banyak jenis namespace dan semua mengisolasi *container*. *Namespaces* tersebut dibuat menggunakan *system call* clone. Suatu *namespace* dapat disambungkan ke *namespace*. Beberapa *namespace* yang digunakan oleh docker adalah PID, Net, IPC, Mnt (*mount*), Uts dan *User*. *Control groups* (cgroups) menyediakan pembatasan (*limitation*) dan *accounting* terhadap sumber daya dari *container*. Dokumentasi Kernel Linux menyatakan bahwa *cgroups* menyediakan mekanisme untuk mengagregasi atau mempartisi himpunan tugas dan semua sub-tugas berikutnya

ke dalam kelompok hirarki sesuai dengan perilakunya. *cgroups* ini dapat dibandingkan dengan perintah *shell* ulimit atau *system call* setrlimit. *cgroups* mengizinkan pembatasan sumberdaya ke suatu grup proses, tidak hanya terhadap proses tunggal. *cgroups* dapat dipecah ke dalam beberapa subsistem seperti CPU, himpunan CPU, blok memory dan I/O.

Cara kerja *cgroups* ialah membagi *resources* ke dalam beberapa kelompok dan menetapkan tugas apa saja yang akan dilakukan dalam kelompok tersebut. Docker menggunakan *cgroups* untuk membatasi *system resources*. Ketika suatu *docker container* dijalankan, maka *cgroups* akan secara otomatis ter-*install* dan membuat direktori subsistem. *Cgroups* dapat dikelola dengan perintah "lscgroup".

Docker menggunakan *client* dan *server*. *Docker client* mengirimkan *request* ke *docker daemon* untuk membangun, mengirimkan, mendistribusikan, dan menjalankan *docker container*. *Docker client* dan *docker daemon* dapat berjalan pada sistem yang sama. Antara *docker client* dan *docker daemon* berkomunikasi via *socket* menggunakan *RESTful API*. Docker menyatukan perangkat lunak dalam file system lengkap yang berisi semua yang diperlukan untuk menjalankan perangkat lunak tersebut. Contohnya: *source code*, paket sistem untuk *runtime*, perangkat sistem, sistem pustaka *software* dan apapun yang dapat diinstal pada *server*. Hal ini menjamin bahwa perangkat lunak akan selalu berjalan, tidak tergantung pada apapun lingkungannya. Pada perkembangan terkini, docker bahkan dapat dijalankan pada sistem operasi Windows maupun MacOS.

2.2 CIA Triad

CIA Triad dan Parkerian Hexad merupakan aturan dasar atau prinsip keamanan dasar dalam menentukan keamanan suatu jaringan atau informasi apakah aman atau tidak. C.I.A Triad adalah kepanjangan dari *Confidentiality*, *Integrity* dan *Availability*, merupakan suatu konsep *framework* yang digagas *The Committee on National Security Systems* (CNSS) dalam memahami model keamanan informasi (*InfoSec*). Konsep ini merupakan standar industri sejak digunakannya *mainframe* dalam teknologi informasi yang menggambarkan 3 karasteristik terpenting yang dipakai dalam pengelolaan sitem informasi, yakni:

1. *Confidentiality*, berkaitan dengan penggunaan hak akses, dimana suatu sistem komputer hanya dapat diakses sesuai dengan hak yang telah ditentukan, kepada siapa dan sejauhmana akses yang diberikan. Merupakan aspek dalam keamanan jaringan yang membatasi akses terhadap informasi, dimana hanya orang-orang yang telah mendapatkan izin yang bisa mengakses informasi tertentu. Hal ini untuk mencegah bocornya informasi ke orang-orang yang tidak bertanggung jawab. Seperti yang kita ketahui, pada masa sekarang ini, informasi merupakan hal yang sangat berharga, contohnya nomor kartu kredit, informasi personal, account bank, dll. Informasi-informasi seperti itu harus dijaga kerahasiaannya agar tidak bisa digunakan dengan sembarangan oleh orang lain. Salah satu komponen penting dalam menjaga *confidentiality* suatu informasi adalah dengan enkripsi. Enkripsi bisa digunakan untuk menjamin bahwa hanya orang yang tepat yang bisa membaca (mendekripsi) informasi yang dikirimkan. Salah satu contoh enkripsi yang cukup sering digunakan adalah SSL/TLS, suatu protokol *security* untuk berkomunikasi lewat internet.

2. *Integrity*, terkait dengan integritas aset suatu sistem informasi dimana proses penyimpanan, pemrosesan dan transmisi data tidak terekspos dan memungkinkan terjadinya campur tangan pihak lain yang dapat merusak atau merubah keaslian data. *Integrity* merujuk kepada tingkat kepercayaan terhadap suatu informasi, kepecayaan dalam hal ini mencakup akurasi dan konsistensi terhadap informasi yang ada. Oleh karena itu perlu adanya proteksi terhadap suatu informasi dari modifikasi oleh pihak-pihak yang tidak diizinkan. Mekanisme proteksi *integrity* dapat dibagi menjadi dua, yakni: mekanisme priventif (kontrol akses untuk menghalangi terjadinya modifikasi data oleh orang luar) dan mekanisme detektif, yang berguna untuk mendeteksi modifikasi yang dilakukan orang luar saat mekanisme priventif gagal melakukan fungsinya.

3. *Availability*, adalah ketersediaan informasi dalam format tertentu tanpa adanya halangan kepada pihak yang telah diberi hak untuk mengaksesnya. Konsep *availability* dari suatu informasi berarti bahwa informasi tersebut selalu tersedia ketika dibutuhkan bagi orang-orang yang memiliki izin terhadap informasi tersebut. Sehingga ketika dibutuhkan oleh user, data/informasi dapat dengan cepat diakses dan digunakan. Salah satu serangan terhadap *availability* suatu informasi yang paling dikenal adalah *Distributed Denial of Service* (DDoS). Tujuan utama dari *DDOS attack* adalah untuk memenuhi resourse yang disediakan untuk user, sehingga user tidak bisa mengakses informasi yang seharusnya bisa didapatkan. Selain itu, faktor kelalaian manusia dapat juga mengakibatkan berkurangnya *availability* dan

secara tidak langsung berdampak pada triad yang lain. Faktor lainnya adalah faktor bencana alam, meskipun jarang terjadi akan tetapi dampak yang diakibatkan kadang lumayan besar. Salah satu cara untuk menjamin *availability* suatu informasi adalah dengan cara *backup*. *Backup* yang dilakukan secara berkala dapat meminimalisir dampak yang ditimbulkan. Sedangkan untuk data-data yang sifatnya sangat penting, perlu adanya suatu *server* cadangan atau skema proteksi lainnya yang menjamin bahwa data-data tersebut akan selalu tersedia meskipun terdapat beberapa gangguan.

2.3 Parkerian Hexad

Parkerian Hexad adalah metode yang menggunakan enam unsur keamanan informasi dimana tiga unsur diantaranya adalah unsur dari konsep CIA TRIAD. Unsur yang ditambahkan dari CIA ialah *Authenticity*, *Possessing* or *Control* dan *Utility*.

1. *Authenticity*, berhubungan dengan cara untuk menyatakan bahwa informasi yang akan diakses betul-betul diakses oleh orang yang berhak. Metode atau cara yang menyatakan data dan informasi diakses oleh orang yang benar-benar berhak. *Authenticity* digunakan untuk meyakinkan orang yang mengakses service dan juga *server* (*web*) yang memberikan *service*. Biasanya dengan menggunakan metode *password* dimana terdapat suatu karakter yang diberikan oleh pengguna ke *server* dan *server* mengenalinya sesuai dengan *policy* yang ada. Mekanisme yang umum digunakan untuk melakukan authentication di sisi pengguna biasanya terkait dengan sesuatu yang dimiliki (misalnya kartu ATM, *chipcard*). Sesuatu yang diketahui (misalnya *user id*, *password*, PIN). Sesuatu yang menjadi bagian dari kita (misalnya sidik jari, iris mata) Satu diantara mekanisme menunjukkan keaslian *server* adalah dengan *digital certificate*. Biasanya situs yang dirujuk adalah https.

2. *Possessing* or *Control*, kepemilikan atau kontrol mengacu pada disposisi fisik media data yang disimpan.

3. *Utility*, informasi harus berguna contoh: Enkripsi data pada *disk* untuk mencegah akses yang tidak sah dan kemudian kehilangan kunci dekripsi data yang terenskripsi sangat berguna sebagai satu diantara bentuk pencegahan terhadap penggunaan data yang tidak sah dan merugikan.

2.4 Setuid

Setuid merupakan singkatan dari *set user id*, adalah suatu jenis file khusus untuk perizinan di Unix dan sistem operasi Linux. Setuid adalah alat keamanan yang memungkinkan pengguna untuk menjalankan program tertentu dengan hak istimewa yang telah ditingkatkan (Hope, 2019).

Ketika izin setuid file yang dapat dieksekusi telah diatur, *user* dapat menjalankan program dengan tingkat akses yang cocok dengan *user* yang memiliki file setuid tersebut. Misalnya, ketika *user* ingin mengubah kata sandi mereka, mereka menjalankan perintah *passwd*. Program *passwd* dimiliki oleh akun *root* dan ditandai sebagai setuid, sehingga *user* untuk sementara waktu akan diberikan akses *root* untuk tujuan mengganti kata sandi mereka.

2.5 Docker Daemon Attack Surface

Docker Daemon berada di antara *docker client* dan *container*. *User* tidak secara langsung berinteraksi dengan *docker daemon*, melainkan berinteraksi melalui *docker client*. *Docker daemon* membutuhkan hak akses *root* karena itu pengguna yang mempunyai hak akses *root* saja yang boleh mengendalikan *docker daemon*. Docker memungkinkan berbagi direktori antara *docker host* dan *guest container*. Hal ini memungkinkan *user* untuk menggunakan docker tanpa membatasi hak akses *container*. Masalahnya di sini adalah bahwa *user* dapat memulai sebuah *container* di mana direktori atau *host* yang akan menjadi direktori pada *docker host* dan *container* tersebut akan mengubah sistem *file host* tanpa batasan apa pun (Jeeva Chelladhurai, Pethuru Raj Chelliah, Sathish Alampalayam Kumar, 2016).

Beberapa contoh dari serangan *docker daemon surface attack* yaitu:

1. *External Attacker*, serangan ini merupakan serangan yang dilakukan dari *user* yang menggunakan docker dan menjalankan *docker images shellshock*. *Shellshock* merupakan *docker images* yang memmpunyai kerentanan sehingga jika dijalankan di sebuah *docker host*, maka serangan ini dapat mengakses host melalui *docker images shellshock*.
2. *Malicious Insider*, serangan ini dilakukan oleh *user* yang berada dalam *docker group*. Misalkan ada *docker container* yang dijalankan oleh *user* yang termasuk dalam *docker group*, lalu *docker container* tersebut ternyata mengandung *dockerfile* yang mengubah *user id* menjadi *root*. Sehingga *user* tersebut dapat menduplikasi *id root* dan seolah-olah menjadi *root.*

3. *Container breakout techniques*, serangan ini dilakukan dengan cara *mounting docker sock* sehingga *attacker* dapat mengakses *host filesystem* dan dapat melihat file apa saja yang terdapat dalam *host*.

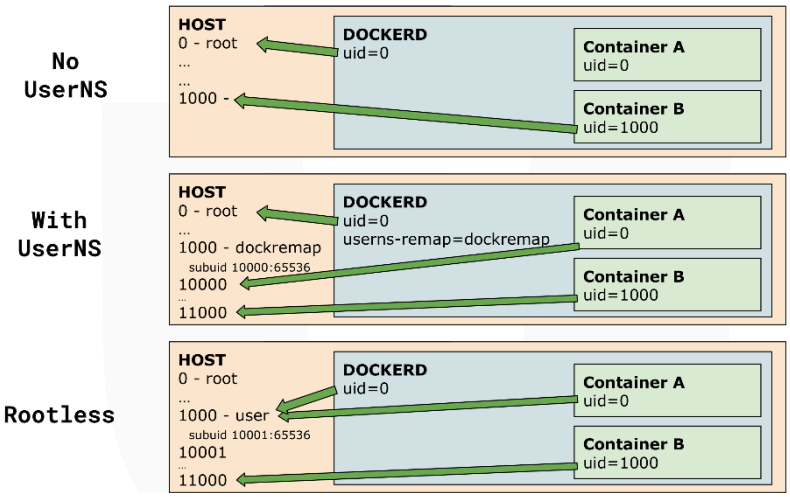
2.6 File /etc/shadow

File /etc/shadow menyimpan kata sandi aktual dalam format yang terenkripsi (lebih seperti *hash password*) untuk akun *user* dengan properti tambahan yang terkait dengan kata sandi *user*. Pada dasarnya, file /etc/shadow ini menyimpan informasi akun *user* secara aman (Understanding the /etc/shadow file, 2019). Semua bidang enkripsi dipisahkan oleh simbol titik dua (:). Bidang ini berisi satu entri per baris untuk setiap *user* yang terdaftar pada file /etc/passwd.

2.7 Rootless Mode

Rootless mode merupakan fitur dari docker yang dapat membuat *user* menjalankan *docker daemon* dan *container* sebagai *non-root user*. *Rootless mode* ini digunakan untuk mengurangi potensi terhadap serangan pada *docker daemon* dan *runtime container*. Selama syaratnya terpenuhi, *rootless mode* tidak membutuhkan *root privileges* dalam proses penginstalasian/pemasangan *docker daemon*. *Rootless mode* diperkenalkan pada versi *docker engine 19.03*. Cara kerja *rootless mode* ini adalah ia akan mengeksekusi *docker daemon* dan *containers* di dalam *user namespace*. Hal ini sangat mirip dengan *userns-remap mode*, perbedaannya adalah pada *rootless mode docker daemon* tidak dijalankan pada *root privileges* (Suda, 2019).

Fitur dari *user namespace* telah ada pada docker untuk waktu yang lama dengan *flags* --userns-remap yang memetakan *user* di dalam *container* ke *range* yang berada di *host*, memberikan keamanan yang lebih baik jika ada *container* yang memiliki akses ke sumber daya eksternal yang sama. *Rootless mode* bekerja dengan cara yang sama, kecuali proses dimana *user* membuat *namespace* terlebih dahulu dan memulai *docker daemon* yang sudah berada di *remapped namespace*.



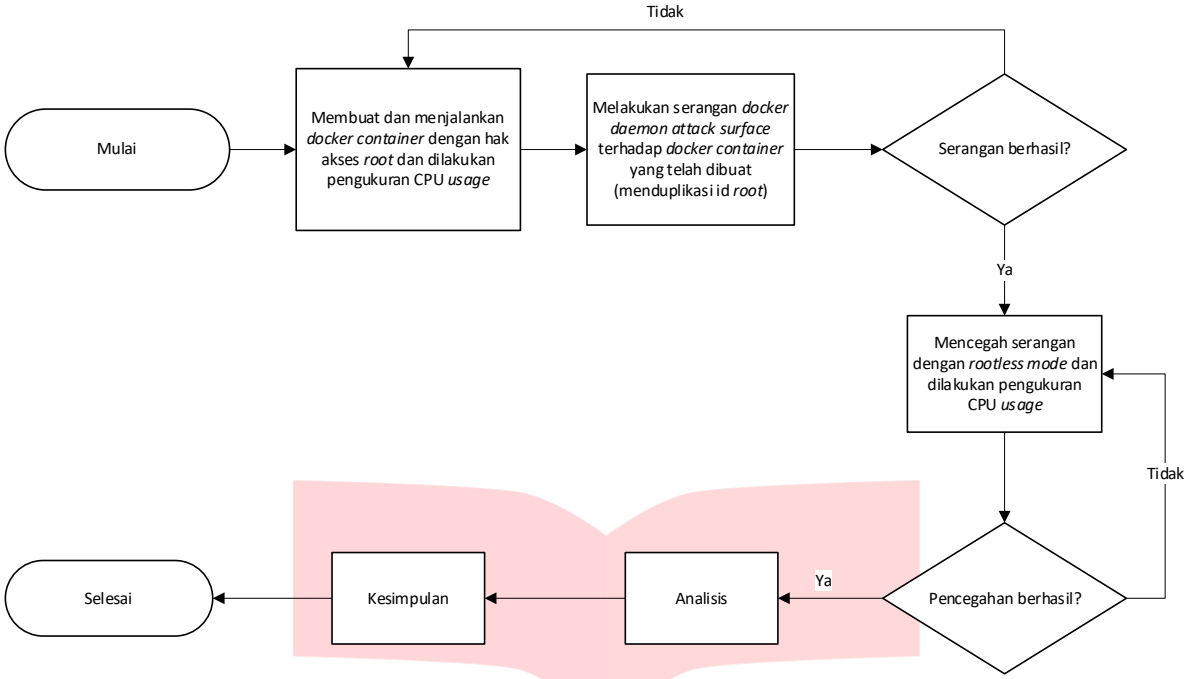
Gambar 1 Cara kerja rootless mode

Daemon dan *container* akan sama-sama menggunakan *user namespace* yang sama yang berbeda dari host. Meskipun linux mengizinkan untuk membuat *user namespaces* tanpa *extended privileges* (tanpa *root*), *namespace* hanya memetakan satu *user* dan karena itu *user* tidak dapat menggunakan *container*. Untuk mengatasinya, *rootless mode* memiliki ketergantungan pada *uidmap package* yang dapat melakukan pemetaan ulang *user*. Binari dalam *uidmap package* menggunakan *bit setuid* sehingga *user* seolah-olah dijalankan dengan *root* secara internal. Untuk merilis *namespace* dan *integrasi* yang berbeda dengan *uidmap*, secara sederhana Akihiro menciptakan sebuah proyek yang disebut *rootlesskit*. *Rootlesskit* juga menangani pengaturan jaringan untuk *rootless containers*.

3. Sistem yang Dibangun

3.1 Alur Proses

Sistem pencegahan serangan *docker daemon attack surface* akan menggunakan *rootless mode* untuk mencegah serangan tersebut. Dan akan dilakukan pengukuran CPU *usage* saat proses penyerangan dan pencegahan. Gambar 1 akan menunjukkan alur proses pada tugas akhir ini.



Gambar 2 Alur Proses

Berikut adalah penjelasan pada Gambar 1:

- Membangun (build) dan menjalankan (*run*) sebuah *container* pada docker yang diakses dengan *root* untuk disimulasikan dalam proses penyerangan. Dalam proses ini juga dilakukan pengukuran CPU *usage.*
- Melakukan serangan yang akan dilaksanakan yaitu menduplikasi id *root* untuk digunakan oleh *user* yang berada dalam *docker group* sehingga *user* yang telah menduplikasi *id root* tersebut dapat mengakses file yang hanya dapat diakses oleh *user* dengan hak akses *root*, contohnya file *shadow* yang terdapat pada foler */etc/shadow*.
- Jika serangan berhasil, maka proses selanjutnya adalah mencegah serangan dengan menggunakan *rootless mode* pada docker. Jika serangan gagal, maka kembali ke proses pembuatan *container* dengan hak akses *root*.
- Pencegahan dilakukan dengan menggunakan *rootless mode*, pengukuran CPU *usage* juga dilakukan. Jika pencegahan berhasil, maka dilanjut ke proses analisis. Jika pencegahan gagal, maka diulang dari proses penyerangan.
- Pada proses analisis, akan dianalisis mengapa serangan gagal dilakukan pada docker yang sedang menggunakan *rootless mode* namun serangan berhasil dilakukan pada docker yang dijalankan dengan hak akses *root* dan hasil pengukuran CPU *usage* juga dianalisis apakah *rootless mode* menambahkan beban kepada CPU atau tidak.
- Proses terakhir adalah proses menyimpulkan apakah *rootless mode* lebih aman dan lebih besar dalam proses penggunaan CPU tersebut daripada menggunakan docker dengan hak akses *root* melalui proses serangan dan pencegahan yang telah dilakukan.

3.2 Penentuan *Software* dan *Hardware*

Pada tugas akhir ini akan digunakan dua jenis perangkat, yaitu perangkat keras dan perangkat lunak. Berikut adalah perangkat-perangkat yang digunakan dalam proses mengerjakan tugas akhir.

3.3.1 Perangkat Keras (*Hardware*)

Berikut adalah perangkat keras (*hardware*) yang akan digunakan untuk menyusun tugas akhir ini:

1. 1 buah laptop yang akan digunakan *user* dan *attacker* untuk melakukan serangan dan pencegahannya.

Tabel 1 *Hardware* yang digunakan

Operating System	*Ubuntu Desktop 18.04 (Bionic Beaver)*
Processor	*Intel i3 4030U 4th Gen*
RAM	*6 GB ddr3*

3.3.2 Perangkat Lunak (*Software*)

Berikut adalah perangkat lunak (*software*) yang akan digunakan untuk menyusun tugas akhir ini:

1. Ubuntu Desktop 18.04 sebagai sistem operasi.
2. Docker untuk membangun dan menjalankan *container*.
3. *GNU Compiler Collections* untuk menjalankan program yang berbahasa C dalam proses penyerangan.

3.4 Proses Penyerangan Serangan dan Pencegahan Serangan Terhadap Docker

Pada saat proses penyerangan dalam tugas akhir ini akan dilakukan penduplikasian id *root*. Id *root* mempunyai nilai nol. Proses penyerangan akan dilakukan dengan menggunakan *user* biasa yang merupakan bagian dari *docker group*.

3.4.1 Proses Penyerangan Serangan Terhadap Docker Yang Dijalankan Dengan Hak Akses *Root*

Pada tahap ini akan dilakukan proses penduplikasian id *root* dan mencoba mengakses file "shadow" yang sebenarnya hanya dapat diakses oleh *super user* (*root*). Pada proses penduplikasian kali ini, docker dijalankan dengan hak akses *root*. Proses yang pertama dilakukan untuk penyerangan ini ialah menyiapkan komponen-komponen penyerangan yaitu :

1. File Dockerfile

File ini berfungsi untuk mengunduh (*pull*) *docker image alpine 3.5* dari *dockerhub* dan menyalin file *root.sh* dan file *rootshell* ke dalam *container* yang akan dijalankan.

2. File rootshell.c

Setelah file *rootshell.c* ini dibuat, file ini harus dijalankan (*run*) menggunakan GCC (*GNU Compiler Collections*) agar dapat digunakan untuk proses penyerangan. Sebenernya file ini tidak sepenuhnya diperlukan, namun sebagian besar *shell* dan program lainnya menolak jika untuk dijalankan sebagai biner setuid. Jadi, file *rootshell.c* ini diperlukan untuk menjalankan biner *setuid* di program atau *shell* yang akan diserang.

3. File root.sh

File *root.sh* menyalin biner *rootshell* ke folder yang dituju (/persist/rootshell) dan mengatur bit setuid yang ada di dalamnya. Dalam file ini terdapat perintah "*chmod* 4777" yang mengartikan bahwa file root.sh mengatur izin hak akses (chmod = *change mode*) sehingga *owner* dapat membaca (*read*), menulis (*write*) dan dapat mengeksekusi (*execute*). *User group* dapat membaca (*read*), menulis (*write*) dan dapat mengeksekusi (*execute*). *User* lainnya (*others*) dapat membaca (*read*), menulis (*write*) dan mengeksekusi (*execute*). File ini merupakan kunci dari proses penyerangan, jika file ini berhasil dijalankan, berarti serangan berhasil dilakukan.

Jika ketiga file tersebut sudah berhasil dibuat maka langkah selanjutnya adalah membuat *docker container* dengan docker yang diakses dengan hak akses *root*. Setelah *docker container* telah berhasil dibuat dan dijalankan. Akhir dari proses penyerangan adalah dijalankannya file /tmp/persist/rootshell. Jika file tersebut dapat dijalankan maka serangan dianggap berhasil. Setelah proses penyerangan dilakukan akan dilakukan uji coba apakah serangan berhasil dilakukan atau tidak. Cara pengujiannya yaitu dengan mengecek *user id* apakah berubah menjadi nol dan *user* dapat mengakses file /etc/shadow. Jika *user id* berubah menjadi nol dan file /etc/shadow telah berhasil diakses maka serangan yang telah dilakukan telah berhasil. Pada saat proses penyerangan berlangsung, dilakukan pengukuran CPU *usage* dengan menggunakan perintah 'docker stats' untuk dibandingkan dengan proses pencegahan serangan dengan *rootless mode*.

3.4.2 Proses Pencegahan Serangan Terhadap Docker Yang Dijalankan Dengan *Rootless Mode*

Pada saat proses pencegahan serangan dalam tugas akhir ini akan dilakukan pencegahan serangan dengan cara penduplikasian id *root*. Id *root* mempunyai nilai nol. Proses pencegahan dilakukan dengan menggunakan *rootless mode* pada docker, akan dilakukan penduplikasian id *root.* Pada proses penduplikasian kali ini docker akan dijalankan dengan *rootless mode*. Tahap selanjutnya semuanya sama seperti dengan apa yang telah dilaksanakan sebelumnya dalam proses penyerangan dengan menggunakan hak akses *root*. Hanya berbeda pada saat penggunaan docker. Jika pada proses penyerangan sebelumnya docker dijalankan dengan hak akses *root*, maka saat ini docker dijalankan dengan *rootless mode*.

Serangan yang akan dicegah dan akan gagal dikarenakan user menggunakan rootless mode pada docker yang cara kerjanya adalah ia akan mengeksekusi *docker daemon* dan *containers* di dalam *user namespace*. Sangat mirip dengan *userns-remap mode*, perbedaannya adalah pada *rootless mode*, *docker daemon* tidak dijalankan pada *root privileges*. *--userns-remap* berfungsi untuk memetakan *user* di dalam *container* ke *range* yang berada di *host*, r*ootless mode* bekerja dengan cara yang sama, kecuali proses dimana *user* membuat *namespace* terlebih dahulu dan memulai *docker daemon* yang sudah berada di *remapped namespace*. *Daemon* dan *container* akan sama-sama menggunakan *user namespace* yang sama namun yang berbeda dari host. Meskipun linux mengizinkan untuk membuat *user namespaces* tanpa *extended privileges* (tanpa *root*), *namespace* hanya memetakan satu *user* dan karena itu *user* tidak dapat menggunakan *container*. Untuk mengatasinya, *rootless mode* memiliki ketergantungan pada *uidmap package* yang dapat melakukan pemetaan ulang *user*.

Pada saat proses pencegahan berlangsung, dilakukan pengukuran CPU *usage* dengan menggunakan perintah 'docker stats' untuk dibandingkan dengan proses penyerangan serangan dengan hak akses *root*. Data CPU *usage* yang telah diukur akan menunjukkan apakah ada penambahan beban pada proses docker yang dijalankan dengan *rootless mode* dibandingkan proses docker yang dijalankan dengan hak akses *root*.

4. Evaluasi

Pada bab ini dibahas mengenai hasil pengujian dan analisis dari proses penyerangan dan proses pencegahan yang telah dilakukan sebelumnya. Terdapat dua pengujian yaitu pengujian serangan menggunakan hak akses root dan pengujian pencegahan serangan menggunakan *rootless mode*. Pengujian dilakukan dengan menduplikasi sebuah id *root* dengan docker yang dijalankan melalui hak akses *root* dan *rootless mode*. Pengujian tersebut dilakukan untuk menunjukkan apakah docker dengan *rootless* mode lebih aman daripada docker yang dijalankan dengan *root*.

4.1 Hasil Pengujian Serangan

```
File  Edit  View  Search  Terminal  Help
reyhanr@mylepy:~$ cd docker2root/
reyhanr@mylepy:~/docker2root$ docker build --rm -t docker2root .
Sending build context to Docker daemon  13.31kB
Step 1/3 : FROM alpine:3.5
 ---> f80194ae2e0c
Step 2/3 : COPY root.sh root.sh
 ---> Using cache
 ---> b2aa101b6822
Step 3/3 : COPY rootshell rootshell
 ---> Using cache
 ---> 2ecae32f817d
Successfully built 2ecae32f817d
Successfully tagged docker2root:latest
reyhanr@mylepy:~/docker2root$ docker run -v /tmp/persist:/persist docker2root:la
test /bin/sh root.sh
reyhanr@mylepy:~/docker2root$ /tmp/persist/rootshell
# id
uid=0(root) gid=1000(reyhanr) groups=1000(reyhanr),4(adm),24(cdrom),27(sudo),30(
dip),46(plugdev),116(lpadmin),126(sambashare),999(docker)
# cat /etc/shadow
root:!:18355:0:99999:7:::
daemon:*:18295:0:99999:7:::
bin:*:18295:0:99999:7:::
sys:*:18295:0:99999:7:::
```

Gambar 3 Output serangan berhasil dilakukan

Dari pengujian serangan yang dilakukan pada subbab 3.4.1 terhadap docker yang diakses dengan hak akses *root*, terlihat pada gambar 2 bahwa *user id (uid)* telah berubah menjadi nol dan *user* dapat mengakses file /etc/shadow. Hal ini mengartikan bahwa docker yang dijalankan dengan hak akses *root* dapat terserang oleh serangan *docker daemon attack surface* yang dilakukan dengan cara menduplikasi *user id* menjadi *id root*. Didapatkan juga hasil pengukuran CPU *usage* yang telah diukur selama proses penyerangan berlangsung.

Tabel 2 Data CPU *usage* (*root*)

Percobaan ke -	CPU *usage* (*root*) dalam %	Memory *usage* (*root*) dalam %
1	0	0.02
2	0	0.02
3	0	0.02
4	0	0.02
5	0	0.02
Rata-rata	0	0.02

Dari tabel 2 dapat dilihat bahwa telah didapatkan data CPU *usage* dengan lima kali percobaan dalam jangka waktu satu menit dan menampilkan rata-rata 0% dan memory (RAM) *usage* hanya 0.02%. Angka yang cukup rendah karena memang pada dasarnya penggunaan docker dalam suatu sistem operasi biasanya tidak terlalu kompleks dan penggunaannya juga diatur oleh *control group* (*cgroups*) sehingga tidak membutuhkan banyak CPU dan memory *usage*. Data tersebut akan dibandingkan dengan proses pencegahan serangan yang dilakukan docker dengan menggunakan *rootless mode* akan menambah beban atau tidak.

4.2 Hasil Pengujian Pencegahan Serangan

```
File  Edit  View  Search  Terminal  Help
 ---> Using cache
 ---> f691dc9c9fef
Successfully built f691dc9c9fef
Successfully tagged docker2root:latest
reyhanr@mylepy:~/docker2root$
reyhanr@mylepy:~/docker2root$ docker run -v /tmp/persist:/persist docker2root:la
test /bin/sh root.sh
reyhanr@mylepy:~/docker2root$
reyhanr@mylepy:~/docker2root$ /tmp/persist/rootshell
$ id
uid=1000(reyhanr) gid=1000(reyhanr) groups=1000(reyhanr),4(adm),24(cdrom),27(sud
o),30(dip),46(plugdev),116(lpadmin),126(sambashare),999(docker)
$
$ cat /etc/shadow
cat: /etc/shadow: Permission denied
$
$ exit
reyhanr@mylepy:~/docker2root$ systemctl --user status docker
● docker.service - Docker Application Container Engine (Rootless)
   Loaded: loaded (/home/reyhanr/.config/systemd/user/docker.service; enabled; v
endor preset: enabled)
   Active: active (running) since Mon 2020-07-13 11:03:32 WIB; 1min 42s ago
     Docs: https://docs.docker.com
 Main PID: 5288 (rootlesskit)
```

Gambar 4 Output serangan gagal (berhasil dicegah) dengan *rootless mode*

Dari pengujian pencegahan serangan yang dilakukan pada subbab 3.4.2 terhadap docker yang diakses dengan *rootless mode*, dapat dilihat pada gambar 3 bahwa *user id* tidak berubah menjadi nol dan *user* tidak dapat mengakses file /etc/shadow. Namun proses pencegahan serangan tersebut tidak mengeluarkan *output* apapun. Hal ini mengartikan bahwa docker yang dijalankan dengan *rootless mode* dapat mencegah serangan *docker daemon attack surface* yang dilakukan dengan cara menduplikasi *user id* menjadi *id root*. *Rootless mode* menjalankan *docker container* tanpa hak akses *root*, hal itu yang membuat serangan gagal. Serangan akan gagal dikarenakan user menggunakan rootless mode pada docker yang cara kerjanya adalah ia akan mengeksekusi *docker daemon* dan *containers* di dalam *user namespace*. Sangat mirip dengan *userns-remap mode*, perbedaannya adalah pada *rootless mode*, *docker daemon* tidak dijalankan pada *root privileges*. *--userns-remap* berfungsi untuk memetakan *user* di dalam *container* ke *range* yang berada di *host*, r*ootless mode* bekerja dengan cara yang sama, kecuali proses dimana *user* membuat *namespace* terlebih dahulu dan memulai *docker daemon* yang sudah berada di *remapped namespace*. *Daemon* dan *container* akan sama-sama menggunakan *user namespace* yang sama namun yang berbeda dari host. Meskipun linux mengizinkan untuk membuat *user namespaces* tanpa *extended privileges* (tanpa *root*), *namespace* hanya memetakan satu *user* dan karena itu *user* tidak dapat menggunakan *container*. Untuk mengatasinya, *rootless mode* memiliki ketergantungan pada *uidmap package* yang dapat melakukan pemetaan ulang *user*.

Maka serangan dapat dijalankan namun tidak berhasil. Sehingga *user* biasa tidak dapat seolah-olah menjadi *root*. Didapatkan juga hasil pengukuran CPU *usage* yang telah diukur selama proses pencegahan serangan berlangsung.

Tabel 3 Data CPU *usage* (*rootless*)

Percobaan ke -	CPU *usage* (*rootless*) dalam %	Memory *usage* (*rootless*) dalam %
1	38.53	25.87
2	38.92	25.45
3	39.28	25.08
4	39.90	25.33
5	40.21	25.14
Rata-rata	39.368	25.374

Dari tabel 3 dapat dilihat bahwa telah didapatkan data CPU *usage* dengan lima kali percobaan dalam jangka waktu satu menit dan menampilkan rata-rata 39.368 % dan memory usage dengan rata-rata 25.374% yang berarti bahwa penggunaan docker dengan *rootless mode* ini lebih besar dibandingkan dengan docker yang diakses melalui hak akses root. Pengukuran CPU *usage* ini merupakan pengukuran sebuah *docker container* yang telah dijalankan oleh *docker* dengan *rootless mode.* Proses yang terukur pada pengukuran ini ialah mengontrol *docker daemon* agar dapat menjalankan *docker container*. Peningkatan CPU dan memory usage yang terjadi dikarenakan *rootless mode* ini belum mendukung *control groups* (*cgroups*) sehingga tidak ada pengaturan dan pembatasan dalam penggunaan CPU dan memory. Dengan penambahan beban sebesar itu merupakan hal yang sepadan dengan keuntungan yang didapatkan dan keuntungannya dengan *rootless mode* ialah dapat mencegah serangan *docker daemon attack surface.* Hal ini dikarenakan docker dengan *rootless mode* mempunyai cara kerja yang berbeda dengan docker dengan hak akses *root*. Cara kerja docker dengan *rootless mode* adalah dengan mengeksekusi *docker daemon* dan *containers* di dalam *user namespace*. Mirip dengan *userns-remap mode* namun perbedaannya adalah *docker daemon* dijalankan tanpa *root.*

4.3 Analisis Hasil Pengujian

Keamanan pada docker merupakan hal yang sangat penting dalam proses penggunaan docker. Dari hasil pengujian serangan dan pencegahan yang telah dilakukan, docker dapat diserang jika dijalankan dengan hak akses *root*. Namun serangan berhasil dicegah dengan menggunakan docker yang dijalankan dengan *rootless mode.* Maka, sebaiknya docker dijalankan menggunakan *rootless mode*.

Dari pengujian yang telah dilakukan, *rootless mode* lebih unggul dibandingkan dengan docker yang diakses dengan *root.* Dengan serangan *docker daemon attack surface* dapat dibuktikan bahwa serangan tersebut dapat dicegah dengan *rootless mode* pada docker. Dengan penambahan beban pada CPU *usage* yang telah dibandingkan sebelumnya yaitu sebesar 39% docker dengan *rootless mode* dapat mencegah serangan *docker daemon attack surface* yang telah dilakukan sebelumnya.

5. Kesimpulan

Berdasarkan hasil pengujian dan analisis yang telah dilakukan terhadap docker, menunjukkan bahwa *rootless mode* pada docker lebih aman daripada docker yang dijalankan dengan hak akses *root*. Sehingga didapatkan hasil kesimpulan sebagai berikut:

1. Dengan menggunakan *rootless mode* pada docker, *user* dapat mencegah serangan *docker daemon attack surface*.
2. *Rootless mode* pada docker dapat mencegah proses penduplikasian id *root* dengan cara menjalankan *docker container* tanpa hak akses *root*, hal itu yang membuat serangan gagal. Karena pada saat menjalankan *docker container* yang didalamnya terdapat file *rootshell* yang merupakan perintah untuk *setuid (set user id)* menjadi id *root* sedangkan dengan *rootless mode* pada docker tidak dijalankan dengan id *root*. Maka serangan gagal sehingga *user* biasa tidak dapat seolah-olah menjadi *root* dan mengakses file yang hanya dapat diakses oleh *root user*.
3. *Rootless mode* menambah beban sebesar 39% pada penggunaan CPU dibandingkan dengan docker yang dijalankan menggunakan hak akses root, sudah dibandingkan dan diukur berdasarkan CPU *usage* pada saat proses penyerangan dan pencegahan serangan dilakukan. Penambahan beban tersebut merupakan konsekuensi yang sepadan dengan keuntungan yang akan didapatkan.

Daftar Pustaka

[1] Bui, T. (2015). Analysis of Docker Security. 7.

[2] *Docker Security*. (n.d.). Retrieved from docs.docker.com: https://docs.docker.com/engine/security/security/

[3] Gallagher, S. (2016). Attack surface of Docker daemon. In *Securing Docker*. Packt Publishing.

[4] Hane, O. (2015). *Build Your Own PaaS with Docker*. Packt.

[5] Hope, C. (2019, 2 8). *Setuid*. Retrieved from computerhope.com: https://www.computerhope.com/jargon/s/setuid.htm#:~:text=Setuid%2C%20which%20stands%20for%20set,certain%20programs%20with%20escalated%20privileges.

[6] Jeeva Chelladhurai, Pethuru Raj Chelliah, Sathish Alampalayam Kumar. (2016). Securing Docker Containers from Denial of Service (DoS) Attacks. *IEEE International Conference on Services Computing*, 4.

[7] Pasknel, V. (2019, June 21). *Attacking Docker Environments*. Retrieved from Morphus Lab: https://morphuslabs.com/attacking-docker-environments-a703fcad2a39

[8] Rendek, L. (2020, May 26). *How to install GCC the C compiler on Ubuntu 18.04 Bionic Beaver Linux*. Retrieved from linuxconfig.com: https://linuxconfig.org/how-to-install-gcc-the-c-compiler-on-ubuntu-18-04-bionic-beaver-linux

[9] *Run the Docker daemon as a non-root user (Rootless mode)*. (n.d.). Retrieved from Docker docs: https://docs.docker.com/engine/security/rootless/

[10] Sari Sultan, I. A. (2019). Container Security : Issues, Challenges, and the Road Ahead. *IEEE Acceess*, 21.

[11] *Securing your System*. (n.d.). Retrieved from runnable.com: https://runnable.com/docker/securing-your-docker-system

[12] Shetty, J. (2017). A State-of-Art Review of Docker Container Security Issues and Solutions. *Research Gate*, 5.

[13] Suda, A. (2019). Hardening Docker Daemon with Rootless Mode. *Docker Con 19*, 54.

[14] *Understanding the /etc/shadow file*. (2019, 12 27). Retrieved from linuxize.com: https://linuxize.com/post/etc-shadow-file/