TOBB EKONOMİ VE TEKNOLOJİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DOCKER GÖRÜNTÜLERİNDE ZAFİYET TESPİTİ VE KAPSAYICILARIN GÜVENLİ ÇALIŞTIRILMASI ÜZERİNE BİR MODEL ÖNERİSİ

YÜKSEK LİSANS TEZİ

Özkan ŞENGÜL

Bilgisayar Mühendisliği Anabilim Dalı
Bilgi Güvenliği

Tez Danışmanı: Prof. Dr. Ali Aydın SELÇUK

ARALIK 2021

TEZ BİLDİRİMİ

Tez içindeki bütün bilgilerin etik davranış ve akademik kurallar çerçevesinde elde edilerek sunulduğunu, alıntı yapılan kaynaklara eksiksiz atıf yapıldığını, referansların tam olarak belirtildiğini ve ayrıca bu tezin TOBB ETÜ Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırlandığını bildiririm.

Özkan ŞENGÜL

ÖZET

Yüksek Lisans Tezi

DOCKER GÖRÜNTÜLERİNDE ZAFİYET TESPİTİ VE KAPSAYICILARIN GÜVENLİ ÇALIŞTIRILMASI ÜZERİNE BİR MODEL ÖNERİSİ

Özkan ŞENGÜL

TOBB Ekonomi ve Teknoloji Üniveritesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı
Bilgi Güvenliği

Danışman: Prof. Dr. Ali Aydın SELÇUK

Tarih: Aralık 2021

Kapsayıcı tabanlı sanallaştırma teknolojileriyle birlikte geleneksel sanallaştırma teknolojilerinin tüm donanımın sanallaştırılması yaklaşımının yerine uygulama ihtiyaçlarına özgü kaynakların kullanıldığı bir yaklaşım benimsenmiştir. Kapsayıcı tabanlı sanallaştırma teknolojilerinin kullanımı ile kolaylıkla oluşturulan, taşınabilen uygulama kapsayıcıları; geleneksel sanallaştırma teknolojilerine kıyasla daha performanslı çalışmakta ve etkin kaynak kullanımı sağlamaktadır.

Docker, aynı işletim sisteminin üzerinde çok sayıda birbirinden yalıtılmış ve bağımsız kapsayıcıların çalışmasına imkân sağlayan kapsayıcı tabanlı sanallaştırma teknolojisidir. Kapsayıcı tabanlı sanallaştırma teknolojisi ile sağlanan kaynakların etkin kullanımı, Docker kapsayıcılarının kullanımına olan rağbeti arttırmakla kalmamış kalmamış, mikro servislerin yönetiminde de etkin bir rol üstlenmiştir. Docker kapsayıcılarının kurumsal ve bireysel kullanımının yaygınlaşması; Docker platformunu saldırganlar için önemli birer hedef haline getirmiştir. Docker Hub deposunda bulunan milyonlarca Docker görüntüsünün milyarlarca kez indirilmesi Docker görüntülerinin güvenli çalıştırılması problemini ortaya çıkarmıştır. Docker platformu üzerindeki kapsayıcıların güvenli bir şekilde çalıştırılması, siber saldırılar sonrasında itibar, mali ve hizmet kayıplarının önüne geçebilecektir.

Docker kapsayıcı güvenliğinin sağlanmasına yönelik statik, dinamik analiz ile zararlı yazılım tespiti başlıklarını içerecek şekilde bir çalışma yapılarak, mevcut açık kaynak araçlar aracılığıyla Docker kapsayıcı güvenliğinin bir bütün olarak nasıl ele alınabileceği ile ilgili bir yöntem öne sürülmüştür. Önerilen yöntem ile Docker kapsayıcılarının statik ve dinamik analizleri yapılarak görüntülerin zararlı yazılım ve zafiyetli paket barındırmadığı doğrulanacak, kapsayıcıya ait kaydedilen ağ trafiği analiz edilerek zararlı görüntüler tespit edilecektir. Ayrıca önerilen yöntem vasıtası ile sistem yöneticisi kontrolü dışında açılan portların tespiti yapılarak, Docker kapsayıcıların zararlı davranış sergilemesinin önüne geçilmesi hedeflenmiştir.

Anahtar Kelimeler: Geleneksel sanallaştırma teknolojisi, Docker, IPS/IDS, Kapsayıcı tabanlı sanallaştırma teknolojisi, Yara rules, Ağ trafiği analizi.

ABSTRACT

Master of Science

VULNERABILITY DETECTION ON DOCKER IMAGES AND A MODEL RECOMMENDATION FOR RUNNING IMAGES SAFELY

Özkan ŞENGÜL

TOBB University of Economics and Technology
Institute of Natural and Applied Sciences
Department of Computer Engineering
Information Security

Supervisor: Prof. Dr. Ali Aydın SELÇUK

Date: December 2021

Along with container-based virtualization technologies, an approach that uses resources specific to application needs has been adopted instead of the traditional virtualization technologies' approach to virtualization of all hardware. Easily created and portable application containers with the use of container-based virtualization technologies works better than conventioanl virtualization technologies and provides efficient resource use.

Docker is a container-based virtualization technology that allows multiple isolated and independent containers to run on the same operating system. Effective use of the resources provided by the container-based virtualization technology has not only increased the demand for the use of Docker containers, but also played an active role in the management of microservices. Expansion of corporate and individual use of Docker containers; It has made the Docker platform an important target for attackers. Billions of downloads of millions of Docker images in the Docker Hub repository have created the problem of safe execution of Docker images. To run containers safely on the Docker platform will prevent possible reputation, money and service losses due to cyber attacks.

A study was conducted to include static, dynamic analysis and malware detection headings to ensure Docker container security, and a method was suggested about how Docker container security could be handled as a whole through available open source tools. With the proposed method, static and dynamic analyzes of Docker containers will be made to verify that the images do not contain malicious software and vulnerable packages, and malicious images will be detected by analyzing the recorded network traffic of the container. In addition, by means of the proposed method, it is aimed to prevent the harmful behavior of Docker containers by detecting the ports opened outside the control of the system administrator.

Keywords: Classic virtualization technology, Docker, IPS/IDS, Container-based virtualization technology, Yara rules, Network traffic analysis.

TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla beni yönlendiren danışman hocam Prof. Dr. Ali Aydın SELÇUK'a, kıymetli tecrübelerinden faydalandığım TOBB Ekonomi ve Teknoloji Üniversitesi Bilgisayar Mühendisliği Bölümü öğretim üyelerine, sektörel tecrübeleriyle katkı sağlayan misafir öğretim görevlilerine, yüksek lisans eğitimi süresince desteğini hiçbir zaman esirgemeyen Emrecan ARDA ile Hasan ÖZKILIÇASLAN'a, canım kızım Eylül'e ve destekleriyle her zaman yanımda olan eşim Demet KAYA ŞENGÜL'e çok teşekkür ederim.

İÇİNDEKİLER

ŞEKİL LİSTESİ

Sayfa

ÇİZELGE LİSTESİ

Sayfa

KISALTMALAR

VM : Sanal Makine
LXC : Linux Kapsayıcı
DEVOPS : Yazılım Geliştirme
IAAS : Hizmet Olarak Altyapı
MAC : Zorunlu Erişim Kontrolü
CCAT : Bulut Kapsayıcı Saldırı Aracı
CVE : Bilinen Zaafiyetler ve Açıklar
DOS : Hizmet Dışı Bırakma
DIVDS : Docker Görüntü Zafiyet Tespit Sistemi
BT : Bilgi Teknolojileri
OS : İşletim Sistemi
VMM : Sanal Makine Yöneticisi
CPU : Merkezi İşlem Birimi
NIST : Ulusal Standartlar ve Teknoloji Enstitüsü
AWS : Amazon Web Hizmetleri
EC2 : Elastik Compute Cloud
OSI : Açık Sistem Ara Bağlantısı
DAAS : Hizmet Olarak Veri
CAAS : Hizmet Olarak İletişim
PAAS : Hizmet Olarak Platform
SAAS : Hizmet Olarak Yazılım
HAAS : Hizmet Olarak Donanım
API : Uygulama Programlama Arayüzü
IPC : İşlemler Arası İletişim
DAC : İsteğe Bağlı Erişim Denetimleri
RBAC : Rol Tabanlı Erişim Kontrolü
CLI : Komut Satırı Arayüzü
JSON : JavaScript Nesne Notasyonu
BTRFS : B-Tree Dosya Sistemi
VFS : Sanal Dosya Sistemi
CNM : Kapsayıcı Ağ Modeli
DNS : Alan İsimlendirme Sistemi
DHCP : Dinamik İstemci Ayarlama Protokolü
ISV : Bağımsız Yazılım Satıcısı
PID : İşlem Kimlik Numarası
MITM : Ortadaki Adam
LSM : Linux Güvenlik Modeli
ARP : Adres Çözümleme Protokolü
FTP : Dosya Aktarım Protokolü
CVSS : Genel Açık Derecelendirme Sistemi
CGI : Ortak Ağ Geçidi Arayüzü

1. GİRİŞ

Sanallaştırma teknolojileri vasıtasıyla bilgi sistem kaynaklarında birbirinden yalıtılmış ortamlar oluşturulabilmektedir. Fiziksel sunucular üzerinde birbirinden yalıtılmış sanal ortamların oluşturulmasıyla kaynakların etkin ve güvenli kullanımı sağlanmaktadır. Fiziksel sunucular üzerinde hizmet veren servisler, artık daha güvenli olan sanal sunucular üzerinde çalıştırılmaya başlamıştır. Fiziksel kaynakların daha etkin kullanılması, ölçeklenebilir olması ve güvenli kullanım ortamı sağlaması sanallaştırma çözümlerine olan talebi arttırmıştır. Artan bu talep artışı, bulut bilişim, mikro servis yaklaşımı ve kapsayıcı tabanlı sanallaştırma teknolojileriyle karşılanabilmektedir. 2020 yılı itibariyle endüstride geleneksel sanallaştırma yaklaşımlarının yerine kapsayıcı tabanlı teknolojilerin kullanımının sağladığı faydalar, kapsayıcı tabanlı çözümleri %40 oranında arttırmıştır. Bununla beraber bilgi teknolojileri hizmet sağlayıcılarının %47'sinin kendi ortamlarında kapsayıcı kullanımını arttırmaları beklenmektedir [1]. Birçok modern bilgi işlem iş yükü; her bir fiziksel makinenin, sanal makineler ve kapsayıcılar olarak küçük bilgi işlem birimine bölündüğü çok kiracılı bulut ortamlarında çalıştırılmaktadır. Donanım veya yazılım düzeyinde soyutlamanın gerçekleştirildiği bulut bilişim teknolojisi, yatırım maliyetlerinin düşmesi, ihtiyaca göre ölçeklenebilmesi ve kaynak israfının önlenmesi gibi temel nedenlerden dolayı, günümüzde pek çok kurum ve kuruluşun altyapısını oluşturmaktadır. Hizmet alınan ve hizmet verilen servislerin neredeyse tamamı bulut bilişim aracılığıyla sağlanmaktadır. Kapsayıcılar kısa vadede sanal makinelerin (Virtual Machine s-VMs) yerini tamamen alamayabilir, ancak aynı miktarda alana on kat daha fazla uygulama sığdırdıkları için sanal makinelere olan talebi şimdiden azaltmışlardır [2]. Ancak kapsayıcı tabanlı sanallaştırma teknolojisinin kullanımı beraberinde bir takım güvenlik problemlerini de getirmiştir.

Kapsayıcı tabanlı sanallaştırma; uygulamaları ve uygulama bağımlılıklarını birbirlerinden yalıtılmış ortamlarda çalıştırmak için kullanılan işletim sisteminin sanallaştırılmasıdır. Kapsayıcı sanallaştırması, tüm kaynakların (işlemci, depolama alanı, ana bellek vb.) sanallaştırılması yerine işletim sisteminin doğrudan

sanallaştırılması yaklaşımına dayanır [3]. Kapsayıcılar; içerisinde uygulamalara ait sistem kitaplıkları ve yapılandırma dosyaları bulunan görüntülerden türetilen, çalışan veya durdurulmuş yalıtılmış ortamlardır. Kapsayıcı teknolojisi Linux sistemlerde LXC (Linux Containers) aracılığı ile uzun zamandır biliniyor olmasına rağmen, yaygın bir şekilde kullanılmamıştır. Docker'ın piyasaya sürülmesiyle birlikte kapsayıcılara olan rağbet katlanarak artmıştır [4]. Docker, 2013 yılında DotCloud adlı bir şirket tarafından geliştirilen gelişmiş bir kapsayıcı motorunun açık kaynaklı bir projesi olup, yazılım geliştirme ve dağıtım aşamalarında çok yüksek bir verimlilik sağlayarak kullanımını zamanla yaygınlaştırmıştır. Artan popülarite ile Docker görüntülerinin oluşturulmasına ve oluşturulan görüntülerin herkes tarafından paylaşımına imkân veren Docker Hub servisi oluşturulmuştur. Docker görüntülerinin Docker Hub üzerinden indirilme sayısı sekiz milyarı aşmıştır [5].

Bulut bilişim sağlayıcıları tarafından Docker kapsayıcıların platform olarak servis çözümünde kullanımı ile geliştiriciler için bir standart haline gelen uygulama geliştirme ve dağıtım (DevOps) süreçlerine sağladığı katkı, Docker kapsayıcıların yaygınlaşmasını sağlamıştır. Docker kapsayıcının üzerinde çalıştığı konak işletim sistemini ve çalışan diğer Docker kapsayıcıları saldırılardan korumak için Docker görüntülerindeki zafiyet ve muhtemel tehditlerin tespit edilmesi gerekmektedir. Kapsayıcılar ve konak işletim sistemi arasında doğrudan iletişim kurulması önemli güvenlik zafiyetleri oluşturmuştur. Bu durum saldırganların, konak işletim sistemine ve birlikte konumlandırılmış olduğu kapsayıcılara erişimini, sanal makinelere olan erişime kıyasla daha kolay hale getirmiştir [6].

Docker kapsayıcıların güvenli bir şekilde yalıtımının sağlanması için Linux çekirdeği güvenlik özellikleri ve güvenlik modülleri kullanılmaktadır. Docker kapsayıcıların bilgisayar kaynaklarını kullanımında ise Linux Cgroup mekanizması devreye girmektedir. Linux çekirdeği tarafından uygulanan süper kullanıcı ile normal kullanıcı arasındaki yetki ve hak farklılıklarının olması Docker kapsayıcılar için bir diğer önemli güvenlik mekanizmasıdır. Tüm bu güvenlik mekanizmalarına rağmen içerden ve dışarıdan başlatılabilecek saldırı vektörleri, kapsayıcının üzerinde çalıştığı konak işletim sistemini ve diğer kapsayıcılardaki uygulamaları olumsuz etkileyebilir [7].

1.1 Problemin Tanımı

Docker platformunun, yazılım geliştiricilerin kaynak kullanımını optimize ederken uygulamaları hızla tasarlamasına, geliştirmesine, test etmesine ve dağıtmasına olanak sağlaması, Docker'a olan rağbeti arttırmakla kalmamış, Google ve Amazon gibi birçok kuruluşun Docker'ı yazılım geliştirme yaşam döngülerine dahil etmelerine neden olmuştur [8]. Böylece kapsayıcılar; kendi içerisinde başka kapsayıcıları, çeşitli depoları ve orkestrasyon araçlarını içeren yüksek düzeyde otomatikleştirilmiş karmaşık bir ekosistemin parçası haline gelmiştir. Docker görüntülerine maksatlı olarak eklenen kötücül yazılımlar, sisteme yetkisiz erişim için bırakılan arka kapılar ve bilgisayar virüsleri Docker kapsayıcıların güvenli kullanımı problemini ortaya çıkarmıştır. Ayrıca Docker kapsayıcıların konak işletim sistemi üzerinde fazla kaynak kullanımı, çalışan diğer kapsayıcıların hizmet verememesine neden olabilir.

Temmuz 2017'de bir saldırgan tarafından Docker Hub üzerinde paylaşılan kötücül bir görüntünün, Docker Hub'dan kaldırılmadan önce beş milyondan fazla indirildiği ve yaklaşık 900 bin dolar değerinde 545 Monero dijital parası elde ettiği rapor edilmiştir [1]. Haziran 2018'de Fortinet ve Kromtech güvenlik şirketleri kripto madenciliği programları içeren on yedi tane Docker görüntüsünün beş milyon kez indirildiğini tespit etmiştir [6]. Bilgi teknolojileri güvenliği ve uyumluluğu alanında lider bir yazılım firması olan Tripwire şirketi; 2019 yılında yüzden fazla çalışanı bulunan şirketlerde görev yapan üç yüzden fazla bilgi güvenliği uzmanının katılımıyla bir "Kapsayıcı Güvenliği Durum Raporu" araştırması gerçekleştirmiştir. Kapsayıcıların bulunduğu ortamların bakımından doğrudan sorumlu olan ve rapora katılan bilgi güvenliği uzmanlarının %94'ü kapsayıcı güvenliği konusunda endişeli olduklarını, %60'ı ise kapsayıcı güvenlik olayı yaşadıklarını bildirmişlerdir [9].

Bu tezin amacı; Docker görüntü ve kapsayıcı güvenliğinin sağlanmasına yönelik iki safhalı analiz çerçevesinin uygulandığı bir model önerisi oluşturmaktır. Önerilen modelin ilk safhasında; Docker görüntülerinin içerisinde bulunan paketlere ilişkin açık kaynak Anchore aracı vasıtasıyla zafiyetli paket tespiti ve açık kaynak ClamAV ile zararlı yazılım tespiti yapılmıştır. Modelin ikinci safhasında; Docker kapsayıcılarına ait ağ trafiği kaydedilerek açık kaynak saldırı tespit sistemi Snort3 aracılığıyla ağ trafiğinde anomali tespiti ile Docker kapsayıcıların port taraması gerçekleştirilmiştir.

Böylece iki safhalı model aracılığıyla Docker görüntü ve kapsayıcı güvenliğine yönelik genel bir analiz çerçevesi oluşturulmuştur.

Tezin geri kalan kısmı şu şekilde organize edilmiştir; Bölüm 2'de, Docker kapsayıcı ve görüntü güvenliği ile ilgili literatür taraması yapılmış, kötü niyetli kişilerin kapsayıcılara yaptıkları saldırılar hakkında bilgi verilmiş ve bu saldırılara karşı geliştirilen mekanizmalar açıklanmıştır. Bölüm 3'te, sanallaştırma teknolojilerine ilişkin bilgiler verilmiş, sanallaştırma teknolojilerinin kullanımı ile oluşan fayda ve sakıncalara yer verilerek geleneksel ve kapsayıcı tabanlı sanallaştırma arasındaki farklar açıklanmıştır. Bölüm 4'te Docker kapsayıcı mimarisi, varsayılan ayarlarda Docker kapsayıcı güvenlik mekanizmaları ve kapsayıcı güvenliğine yönelik tehditler açıklanmıştır. Bölüm 5'te, Docker güvenliğinin sağlanmasına yönelik; statik ve dinamik analiz olmak üzere iki safhadan oluşan bir model tasarımı yapılmış, statik analiz safhasında yapılan zafiyetli paket taraması ve imza tabanlı anti virüs taramasına ilişkin bilgiler verilmiştir. Bölüm 5'te önerilen modelin dinamik analiz safhasında; kapsayıcının belli bir süre çalıştırılarak port taraması yapılmış ve ağ trafiğinin analizi sonucu elde edilen bilgiler paylaşılmıştır. Bölüm 6'da Docker kapsayıcı güvenliğinin sağlanması için oluşturulan modelin işlerliği değerlendirilmiştir.

2. LİTERATÜR TARAMASI

Sanallaştırma teknolojileri bilgi ve iletişim teknolojileri endüstrisinin bilgi işlem kaynaklarının daha iyi ve etkin kullanımını sağlamıştır. Kapsayıcı tabanlı sanallaştırma teknolojisine olan eğilim, sanallaştırma mimarisindeki daha küçük boyutlu görüntüler ve bu görüntülerin çalışması ile konak işletim sistemi üzerinde daha az yük oluşturması gibi gelişmelerle birlikte artmıştır. Bu alanda yapılan ilk akademik çalışmalar, genel olarak geleneksel sanallaştırma teknolojileri ile kapsayıcı tabanlı sanallaştırma teknolojileri arasında performans kıyaslamaları üzerine yoğunlaşmıştır. Kapsayıcı teknolojilerine yönelik yapılan daha sonraki çalışmalarda, kapsayıcı kullanımında en büyük endişe kaynağı olan güvenlik hususlarına ağırlık verilmiştir.

Bu bölümde geleneksel sanallaştırma teknolojileri Docker kapsayıcı güvenlik mekanizmaları, Docker görüntülerine ait statik analiz çalışmaları, Docker kapsayıcı güvenliğine yönelik saldırı ve tehditler ile Docker kapsayıcı güvenliğinin sağlanmasına yönelik araştırmalar incelenmiştir.

Hizmet olarak altyapının (IaaS) bulut hizmetinde sistem oluşturma, kaynak sağlama ve çoklu kiracılık (multi-tenancy) için oldukça önemli bir gelişme olarak kabul edilmesi, bu hizmet türünün temelini teşkil eden geleneksel sanallaştırma teknolojilerinin yoğunlukla kullanımının önünü açmıştır. Sanallaştırılmış kaynaklar, bulut bilişimde kaynakların etkin kullanımı, maliyet ve verimlilik gibi sorunların çözülmesinde ana rolü üstlenmektedir [10].

Linux kapsayıcılar ile kaynakların daha verimli bir şekilde dağıtımı mümkündür. Herhangi bir Hyper-V veya VMware sanallaştırma çözümlerinde, maruz kalınan ek yük nedeniyle, ondan fazla sanal makineyi çalıştırmak oldukça zordur. Bu sorun kapsayıcılar ile büyük ölçüde çözülmüş gözükmektedir. Kapsayıcılar, yalnızca hizmetler veya uygulamalar için gerekli olan kaynakları kullanır. Bu nedenle, kaynakları düşük bir makinede, elliden fazla kapsayıcının çalışması mümkün olabilmektedir [11].

Shu ve diğerleri [12] tarafından yapılan çalışmada, Docker Hub platformunda bulunan hem resmî hem de topluluk görüntülerini otomatik olarak keşfeden, indiren ve analiz eden, ölçeklenebilir bir Docker görüntü güvenlik açığı analizi (DIVA) çerçevesi oluşturulmuştur. Oluşturulan DIVA çerçevesi kullanılarak 356.218 görüntü analiz edilmiştir. Analiz neticesinde hem resmî hem de topluluk görüntülerinin tüm sürümleri üzerinde ortalama olarak yüz seksenden fazla güvenlik açığı olduğu tespit edilmiştir. Ayrıca Docker Hub platformunda yer alan pek çok görüntünün yüzlerce gündür güncellenmediği, varolan güvenlik açıklıklarının da ana görüntülerden alt görüntülere yayıldığı tespit edilmiştir. Yapılan çalışma ile elde edilen bulgular sonucunda, Docker görüntülerine güvenlik güncellemeleri uygulamak için daha otomatik ve sistematik yöntemlere güçlü bir ihtiyaç olduğu ortaya konulmakta ve DIVA Docker görüntü analiz çerçevesinin, bu tür otomatik güvenlik güncellemeleri için iyi bir temel sağladığı vurgulanmıştır.

Babak ve diğerleri [13] tarafından yapılan araştırmada hız, kolay taşınabilirlik, ölçeklenebilirlik, kolay kurulum ve kaynakların etkin kullanımı Docker kapsayıcılar tarafından sağlanan avantajlar olarak ön plana çıkmaktadır. Bu çalışmada Docker kapsayıcı mimarisinde konak işletim sistemine ait tüm kaynakların sanallaştırmasının yapılmaması, Docker'ın eski makinelerde çalışmaması, yalnızca 64-bit makineler tarafından desteklenmesi, güvenlik sorunlarının önemli bir başlık olarak değerlendirilmesi gerektiği, akademik ve bilimsel çevre tarafından tam olarak kabul görmeme ihtimali Docker teknolojisinin dezavantajları olarak nitelendirilmiştir.

Sharma ve diğerleri [11] tarafından sanal makine ve kapsayıcıların geniş ölçekli kıyaslaması yapılmıştır. Bu çalışmada sanal makineler ve kapsayıcılar arasındaki farkların performanslarında, yönetilebilirliklerinde ve kullanım durumlarında kendini gösterdikleri ortaya konulmuştur. Donanım sanallaştırmasının güçlü kaynak yalıtımı nedeniyle, sanal makine ortamlarında çoklu kiracılık oldukça yaygındır. Kapsayıcıların konak işletim sistem çekirdeğini paylaşıyor olmaları yalıtım eksikliği meydana getirmektedir. Kapsayıcılar tarafından sağlanan yalıtımının daha zayıf olması nedeniyle, çok kiracılı sistemin uygulanışı riskli kabul edilmektedir. Tanımlanmış kaynak sınırlarına sahip sanal makinelerin aksine kapsayıcılar birbirlerinin kaynaklarını kullanmada esnek bir yapıya sahiptirler.

Tomar ve diğerleri [14] tarafından yapılan çalışmada Docker kapsayıcılara yönelik tehdit modellemesi ve saldırı taksonomisi yapılmıştır. Docker kapsayıcılara yönelik

oluşturulan tehdit modellemesi; uygulamadan kapsayıcıya olan saldırılar, kapsayıcıdan bir diğer kapsayıcıya olan saldırılar, kapsayıcıdan konak işletim sistemine yönelik saldırılar, konak işletim sisteminden kapsayıcıya saldırılar, kapsayıcıdan Docker platformuna yönelik saldırılar, konak işletim sisteminden Docker platformuna yönelik saldırılar ile kullanıcıdan Docker platformuna yönelik saldırılar olmak üzere yedi ana başlık altında yapılmıştır. Yapılan çalışmada kapsayıcılara yönelik saldırılar; zararlı yazılım, kapsayıcı kaçış saldırısı, ARP kandırmaca (ARP spoofing) ve MAC adres taşması (MAC Flooding) saldırıları, ortadaki adam saldırısı (Man-in-the-Middle), yetkisiz erişim, zehirli görüntüler (poisoned images), eski yazılım, bulut kapsayıcı saldırı aracı (CCAT), zararlı kod enjeksiyonu, çekirdek istismarı (Kernel exploit), kripto madencilik zararlısı (cryptojacking), kurcalama (tampering) ve gereksiz servislere yönelik saldırılar olarak sınıflandırılmıştır.

Peiyu ve diğerleri [15] tarafından yapılan çalışmada Docker Hub servisi üzerinden 2.227.244 Docker görüntüsü toplanmış ve bu servise ilişkin güvenlik riskleri ortaya konmaya çalışılmıştır. Docker Hub servisinin popülerliğinin birçok yüksek profilli saldırıyı da beraberinde getirdiğine dikkat çekilmiştir. Bir araştırma enstitüsü 13 Haziran 2018'de, Docker Hub üzerinde bulunan on yedi adet zararlı Docker görüntüsünün kripto madencilik suçlularına otuz günde 90.000 dolar kazandırdığını bildirmiştir. Yapılan çalışmada, çalıştırılan komut parametrelerinin güvenlik riskleri, kötü amaçlı çalıştırılan programlar ile görüntü içerisinde yer alan paketlerin Common Vulnerabilities and Exposures (CVE) güvenlik açıklıkları incelenmiştir. Hassas parametreler ile komut çalıştırılması, Docker Hub platformunda hizmet dışı bırakma saldırısı (DoS) ve konak işletim sistemine sızılması dahil olmak üzere ciddi güvenlik riskleri oluşturmaktadır. Kötücül görüntülerin, Docker Hub'dan yaygın olarak indirilen görüntüler arasına gizlendiği, kötücül görüntü davranışlarının uzaktan yürütme kontrolü ve kötü niyetli kripto madenciliği olduğu bu çalışmada belirtilmiştir. Docker görüntülerinde bulunan paketlere ait güvenlik açığı yamalarının önemli ölçüde geciktiği hatta göz ardı edildiği gözlemlenmiştir. Docker Hub üzerinde bulunan görüntülerin büyük bir kısmının yamalanmamış güvenlik açıkları içerdiğini de belirten çalışmada, bazı sıra dışı durumlarda tek bir görüntüde 7.500'e kadar güvenlik açığı tespiti yapılmıştır.

Wenhao ve diğerleri [16] tarafından yapılan çalışmada Docker kapsayıcı güvenlik açıklıkları; dosya sistemi yalıtımı, proses ve iletişim yalıtımı, cihaz yönetimi ve ana

makine kaynak sınırlaması ağ yalıtımı ve görüntü aktarımı olmak üzere dört başlık altında incelenmiştir. Docker kapsayıcılarda dosya sistemi arasındaki etkileşimin önlenmesi için dosyalar farklı ad alanlarında (namespaces) depolanmaktadır. Ancak ana makine üzerinde kapsayıcıların kendilerine tahsis edilen dosya alanı üzerinde okuma ve yazma izinlerine sahip olması ve bu kapsayıcıların kök kullanıcı izinleri ile çalıştırılması durumunda, çalıştıkları dosya alanı içerisinde yer alan bilgilerin sızma ihtimalini büyük ölçüde arttırmaktadır.

Kwon S. ve Lee J.H [17] tarafından yapılan çalışmada, Docker ve Kubernetes'in kullanıma sunulmasıyla kapsayıcı ortamı DevOps ile birlikte hızla gelişmeye başlamıştır. DIRTY COW (CVE-2016-5195) ve RunC Container Escape (CVE-2019-5736) gibi kapsayıcı ortamlarında yeni güvenlik açıklarının keşfedilmesi ve kapsayıcı kullanımındaki artış, kullanıcıların kapsayıcı güvenliğine yönelik endişelerini arttırdığı belirtilmiştir. Ayrıca Şubat 2018'de, Amazon'un AWS ortamında bulunan Tesla'nın Kubernetes ortamının saldırıya uğramasının ardından kapsayıcı ortamlarda kripto para madenciliği yapan sistemler için önemli bir sorun haline geldiğinden bahsedilmiştir. Yapılan çalışmada Docker görüntülerinin, güvenlik zafiyet taraması yapılmadan paylaşıldığında, güvenli olmayan Docker görüntülerinin dağıtılabileceğini, böylece Docker tabanlı uygulama geliştirme ortamlarının kolayca istismar edilebileceği ifade edilmiştir. Docker görüntülerini bir Docker görüntü havuzuna yüklerken veya indirirken görüntülerin güvenliğini kontrol eden Docker görüntü zafiyet tespit sistemini (Docker Image Vulnerability Diagnostic System DIVDS) önermişlerdir. Ayrıca Docker görüntüleri üzerinde Clair açık kaynak aracıyla statik bir tarama gerçekleştirilmiş ve tarama sonucuna göre bir puanlama yapılmıştır. Bu model ile hesaplanan puanlar, Docker görüntülerinin yüklenmesine veya indirilmesine ilişkin karar vermek amacıyla kullanılmıştır.

Zerouali ve diğerleri [18] tarafından yapılan çalışmada, Docker görüntüleri üzerinde bulunan güncellenmemiş paketler için statik ve dinamik analizlerin birlikte kullanıldığı bir model sunulmuştur. Bu çalışmada Docker Hub deposunda 1.5 milyondan fazla görüntü olduğu ve Docker görüntülerinin zararlı bir şekilde etkilenip etkilenmediğinin araştırılması yerine görüntülerin ne kadar eski olduğuna odaklanılmasının güvenliğin sağlanmasında daha önemli olacağı belirtilmiştir. Docker görüntüsünde çalıştırılacak olan servise ait kodlar güncellense bile görüntünün kendisinin güncellenmediğine dikkat çekilmiştir. Taranan Docker görüntülerinin

yarısından fazlasının dört aydır güncellenmediği ve Docker görüntüleri üzerinde yer alan her beş paketten birinin güncel olmadığı tespit edilmiştir. Yapılan çalışma ile Docker görüntülerinin ne kadar sıklıkla güncellendiği, görüntüde yer alan paketlerin nasıl güncelleme dışı kaldığı, güncellenmeyen paketlerin Docker kapsayıcılar üzerinde ne gibi açıklıklara neden olduğu ve ortaya çıkan zafiyetli paketlerin güncellenmeden Docker Hub depolarında ne kadar süreyle kaldığı gibi sorulara cevap verilmeye çalışılmıştır. Tüm bu bulgularla Zerouli güvenlik açıklıklarının neredeyse yarısının düzeltilmediği ve birçok kapsayıcı üzerinde yüksek önem derecesine sahip güvenlik açıklığı olduğu sonucuna varmıştır.

Kelly ve diğerleri [8] tarafından yapılan çalışmada, Docker görüntülerine yönelik statik ve dinamik analizi içeren bir model geliştirilmiştir. Model ile analiz edilen kötücül görüntülerin genellikle komuta ve kontrol sunucularıyla iletişime geçmek için ssh tünelleri oluşturan bash betikleri içerdiği, ayrıca kötücül görüntülerin ilave ikili dosyaları indirmeye ve kabuk kod yüklemeye çalıştıkları tespit edilmiştir.

3. SANALLAŞTIRMA TEKNOLOJİLERİ

Kurum, kuruluş ve kullanıcıların iş ve hizmetler kapsamındaki ihtiyaçları hızla artmaya devam etmekte olup bu ihtiyaçların otomotize edilerek karşılanması Bilgi Teknolojileri (BT) tarafından sağlanmaktadır. İhtiyaçların hızla artması ve modern dünyanın hızla büyümesi kaynakların etkin olarak kullanımını zorunlu hale getirmiştir. Fiziksel sunucu ve üzerinde koşturulacak olan servis ve uygulama sayıları arasındaki arz ve talep dengesi, sanallaştırma teknolojilerinin kullanılmadığı durumlarda, fiziksel sunucuların sayısal anlamda az olması nedeniyle bozulmuştur. Ortaya çıkan sanallaştırma teknolojisi ile bu denge yeniden kurulmaya çalışılmıştır. Sanallaştırmayı; fiziksel bir kaynağın mantıksal yapılara ayrılması ve ayrılan bu mantıksal yapıların farklı bir fiziksel kaynak gibi gösterilerek bilgisayar sisteminin sanal bir örneğinin çalıştırılması olarak tanımlamak mümkündür. Sanallaştırma, tek bir fiziki bilgisayarın donanım öğelerinin (işlemciler, bellek, depolama vb.), sanal makine olarak adlandırılan birden çok sanal bilgisayara bölünmesine ve fiziki bilgisayar donanımı üzerinde bir soyutlama katmanının oluşturulmasına imkân sağlayan bir yazılım aracılığı ile gerçekleştirilmektedir. Bu teknolojiyle, her sanal makine kendi işletim sistemini (OS) çalıştırır ve fiziki bilgisayar donanımının yalnızca bir bölümünde çalışarak bağımsız bir bilgisayarmış gibi davranır. Günümüzde sanallaştırma, kurumsal BT mimarisinin ve bulut bilişim teknolojilerinin merkezinde yer almaktadır. Bulut servis sağlayıcıları, sanallaştırma teknolojileri sayesinde mevcut fiziksel bilgisayar donanımlarını sanallaştırarak, kullanıcılara ihtiyaç duydukları bilgi işlem kaynaklarını satın alma ve iş yükleri arttıkça bu kaynakları arttırma imkânı sağlayan bir mimari sunmaktadırlar. Sanallaştırma teknolojileri, birden çok sanallaştırılmış uygulama örneğini tek bir fiziksel sunucu üzerinde çoklu kiracılık (multi-tenancy) mantığı ile çalıştırabilmektedir.

Veri merkezi ortamlarında; ilki geleneksel sanallaştırma olarak adlandırılan, donanım düzeyinde ihtiyaç duyulan her birimin sanallaştırılması yaklaşımına dayalı hipervizör veya sanal makina monitörü ile tüm bilgisayar bileşenlerinin emüle edilmeksizin

işletim sisteminin sanallaştırılması yaklaşımına dayalı kapsayıcı tabanlı sanallaştırma teknolojisi olarak adlandırılan iki farklı sanallaştırma teknolojisi kullanılmaktadır.

3.1 Hipervizör Tabanlı Sanallaştırma

Sanallaştırma fikri 1960'ların sonu ve 1970'lerin başında IBM şirketi tarafından zaman paylaşımı (time-sharing) teknolojisini geliştirmeye çalışması ile başlamıştır. Zaman paylaşımı teknolojisini hem kullanıcıların hem de paylaştıkları kaynakların verimliliğini arttırmayı amaçlayan birden fazla uygulamanın birden fazla kullanıcı tarafından kullanılması olarak tanımlamak mümkündür. Kullanıcılar tarafından hissedilmesi mümkün olmayan nano saniye gibi oldukça kısa süreler sayesinde kullanıcı fiziksel kaynağın tamamiyle kendisine ayrıldığını düşünmektedir [19].

Uygulamaların birbirinden yalıtılması için önerilen ilk yöntem, her uygulama için ayrı bir fiziksel sistemin kullanılması düşüncesi olmuştur. Bu öneri çok pahalı bir çözüm olmasının yanı sıra zaman paylaşımının kullanılmaması nedeniyle sistem kaynaklarının israfına yol açmıştır [20].

Sanal makine vasıtasıyla uygulamaların yalıtılması yirmiye bir oranında performans düşüklüğüne neden olmuştur. Bu nedenle araştırmacılar 1960'ların sonlarında simülasyon yazılımının performansını iyileştirmeye odaklanmışlardır. IBM tarafından dönemin en iyi bilinen sanal makine monitörü olan IBM virtual monitör/370 (VMM/370) üretilmiştir [20].

Sanallaştırma teknolojisi 1970'lerde genişlemeye devam etmişse de 1980 ve 1990'lı yıllarda donanım fiyatlarındaki düşüş ve kişisel bilgisayarların gittikçe yaygınlaşması gelişimin yavaşlamasına sebep olmuştur. 1970'lerde bilgisayar mimarileri sanallaştırma göz önünde bulundurularak geliştirilirken ana bilgisayarların yerini kişisel bilgisayarların alması ile VMM'ler için gerekli donanım desteğinin sunulmadığı bilgisayar mimarileri ortaya çıkmıştır. VMM'ler 1980'lerin sonunda tarih olmuştur [19].

VMM'ler, 1990'larda Stanford Üniversitesi'ndeki araştırmacılar tarafından ticari donanım için sanallaştırma teknolojisinin ilk tedarikçisi olan VMware Inc.'in doğmasına yol açan bir araştırma projesiyle yeniden gün ışığına çıkarılmıştır. Yeniden yaygınlaşmaya başlaması SWsoft, XenSource, Microsoft ve Oracle gibi pazar liderlerinin de bu alanda çalışmalar yapmasına neden olmuştur. Sanallaştırma

teknolojileri yalnızca lider yazılım üreticilerinin çalışmalarında değil aynı zamanda Intel ve AMD gibi donanım üreticilerinin ürünlerinde de gerekli karşılığı ve desteği bulmuştur [19].

3.1.1 Hipervizör Tabanlı Sanallaştırma Mimarisi ve Çeşitleri

Hipervizör veya VMM birkaç sanal makinenin fiziksel bir makinede çalışmasına izin veren bir yazılım katmanı olup, bugün herkes tarafından kullanılan bulut teknolojisinin temelini oluşturmaktadır. Hipervizör tabanlı sanallaştırma ile bir bilgisayarın tam olarak taklit edilmesi mümkün olup, akıllı telefon gibi diğer cihaz türleri ile farklı CPU (Central Process Unit) mimarilerine sahip sistemler de emüle edilebilmektedir. Hipervizör tabanlı sanallaştırma sayesinde uygulama geliştiriciler uygulamanın çalıştırılacağı fiziksel cihaz erişimine ihtiyaç duymaksızın kendi geliştirme ortamlarında uygulamalarını test edebilirler. Ayrıca hipervizör tabanlı sanallaştırmayla bazı kullanıcılar tarafından ana işletim sisteminde çalışmayan özel bir yazılımın çalıştırılması gerektiğinde, ana işletim sisteminden bağımsız bir işletim sisteminin oluşturulması oldukça kolaylaşmıştır. Hipervizör tabanlı sanallaştırma Tip 1 hipervizör ve Tip 2 hipervizör olmak üzere iki farklı şekilde sınıflandırılmaktadır [21].

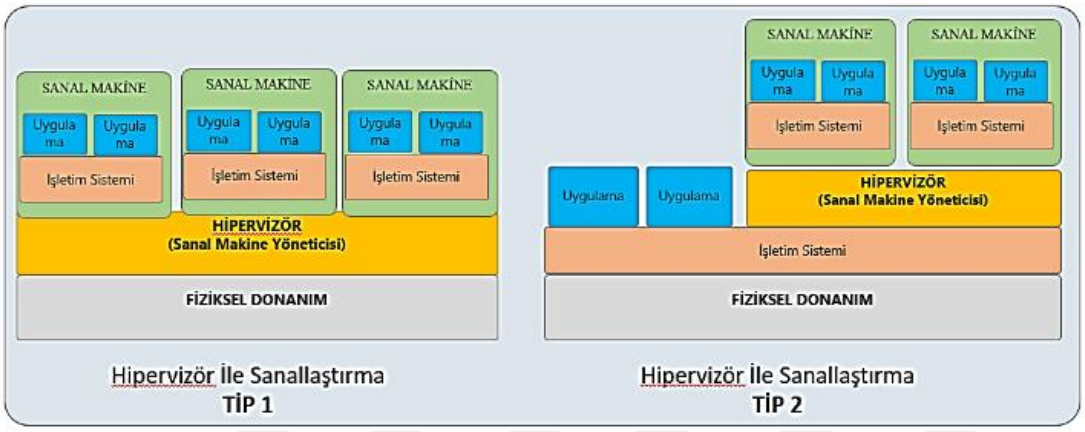
3.1.2 Tip 1 Hipervizör

Tip 1 hipervizör donanımı kontrol etmek ve konuk işletim sistemlerini izlemek için doğrudan ana bilgisayarın donanımında çalışan, çalışması için başka bir işletim sisteminin yüklenmesine ihtiyaç duymayan yazılım sistemleridir. Citrix/Xen Server, VMware ESXi ve Microsoft Hyper-V bu tip sanallaştırmalara örnek teşkil etmektedir. Tip 1 hipervizörlerin temel donanıma doğrudan erişiyor olması, Tip 1 hipervizörleri kurumsal bilgi işlem için en iyi performans gösteren ve en verimli kullanıma sahip hipervizör tabanlı sanallaştırma şekli yapmaktadır [22].

Etkili ve iyi performans gösteren bu hipervizörler oldukça güvenli bir yapıya da sahiptirler. Güvenli olmalarının temelini ise işletim sistemlerine özgü kusurların ve güvenlik açıklıklarının Tip 1 hipervizörlerde olmamasıdır. Yazılımın altında herhangi bir işletim sisteminin olmaması, Tip 1 hipervizör üzerine kurulan birbirinden yalıtılmış her bir sanal makineye yapılacak olan saldırı yüzeyini azaltmaktadır [23].

3.1.3 Tip 2 Hipervizör

Tip 1 ve Tip 2 hipervizörleri arasındaki temel fark, Tip 2 hipervizörlerin bir uygulama gibi mevcut olan bir işletim sistemi üzerine kurulmasıdır. Tip 2 hipervizörler fiziksel kaynaklara konak işletim sistemi vasıtasıyla erişim sağlarlar. Tip 2 hipervizörlerin mevcut işletim sisteminin üzerine bir uygulama gibi kuruluyor olmaları nedeniyle kişisel kullanımda ve test ortamı oluşturulmasında kullanılmaktadırlar. Oracle Virtualbox, Hyper-V ve Vmware Workstation bu tip sanallaştırmalara örnek teşkil etmektedir. Hipervizör tabanlı sanallaştırma tipleri Şekil 3.1'de gösterilmiştir.



Şekil 3.1 : Hipervizör tabanlı sanallaştırma tipleri [24]

3.2 Kapsayıcı Tabanlı Sanallaştırma Teknolojisi

Kapsayıcı tabanlı sanallaştırma, proseslerin yalıtılmış bir ortamda çalıştırılabilmesi için çekirdek özelliklerini kullanır. Ayrıca hipervizör tabanlı sanallaştırmayla oluşturulan sanal makinelerde olduğu gibi tüm donanımın sanallaştırılması yerine sadece ihtiyaç duyulan miktarda fiziksel makine donanımı kullanılır. Kapsayıcılarda çalışan yazılım fiziksel makine çekirdeği ile doğrudan iletişim kurar ve fiziksel makinenin işletim sistemi ve CPU mimarisi üzerinde çalışır. Donanımın taklit edilmemesi ve tam bir işletim sistemi başlatmak zorunda olunmaması, kapsayıcıların birkaç milisaniye içinde başlamasına imkan sağlar [21].

Kapsayıcılar üzerinde çalıştırılacak olan uygulamaların birbirlerinden yalıtılmış bir ortamda çalıştırılması işlemi, her sanal ortam örneği için soyutlanan kullanıcı isim uzayları (user-namespace) vasıtasıyla yapılmaktadır. Bu nedenle kapsayıcı tabanlı sanallaştırma teknolojisi genellikle işletim sistemi düzeyinde sanallaştırma olarak da bilinmektedir.

Linux-VServer, OpenVZ, LXC, Solaris kapsayıcıları, FreeBSD Jails, Kubernetes ve Docker işletim sistemi düzeyinde sanallaştırma yapan teknolojilere örnek olarak verilebilir.

3.2.1 Kapsayıcı Tabanlı Sanallaştırma Teknolojisi Tarihsel Gelişimi

1990'ların ortasında, POSIX (The Portable Operating System Interface) standartları projesinin güvenlik çalışma grubu, POSIX.1 standardına POSIX 1003.1e "yetenekler" (capabilities) adı verilen bir özellik eklemiştir. POSIX yeteneklerinin uygulama ayrıntıları, ilk yetenek sistemlerinden tamamen farklı olsa da kavramsal düzeyde benzerlikler göstermekteydi. POSIX yetenekleri, bir prosesin belirli eylemleri gerçekleştirmesine izin verilip verilmediğini belirleyen bir işlemci dosyasıyla ilişkili bayrak kümesidir. Prosesler kendi yeteneklerinin bir alt kümesine sahip olan bir prosesi kendi alt prosesi olarak çalıştırabilmektedir. Bununla birlikte yeni POSIX yetenekleri kendinden önceki yetenek sistemlerinde güvenli yalıtımın önemli unsurları olan küçük erişim alanları (small access domains) ve ayrıcalıksız varsayılanlar (no-privilege defaults) kavramlarını benimsememiştir. POSIX 1.e taslağı 1998'te kabul sürecinden geri çekilmiş ve resmi olarak hiçbir zaman standart olarak kabul edilmemesine rağmen 1999'da Linux çekirdeğine eklenen yetenekler özelliğinin temelini oluşturmuştur [19].

Modern kapsayıcı teknolojilerinin uygulanmasında ad alanları (namespaces) ve kaynak kullanım kontrolleri bir diğer önemli kilometre taşını oluşturmaktadır. 2000 yılında FreeBSD tarafından dosya sistemi ad alanları, prosesleri ve ağ kaynaklarını chroot aracılığıyla yalıtan jails adı verilen bir yapı oluşturulmuştur. Jails vasıtasıyla oluşturulan bu yapıyla, prosesler jails içerisinde bulundukları sürece kök kullanıcı izinleri ile çalıştırılabilmekte, ancak jails dışındaysa hiçbir işlem yapmalarına müsaade edilmemektedir. Jails, işletim sisteminin belirli yalıtılmış alanlarına erişim izni verme yöntemidir. Tüm yalıtım çekirdek düzeyinde gerçekleşmekte ve kullanıcılar jails sayesinde yalnızca görmeleri gereken bileşenleri görmektedirler [19].

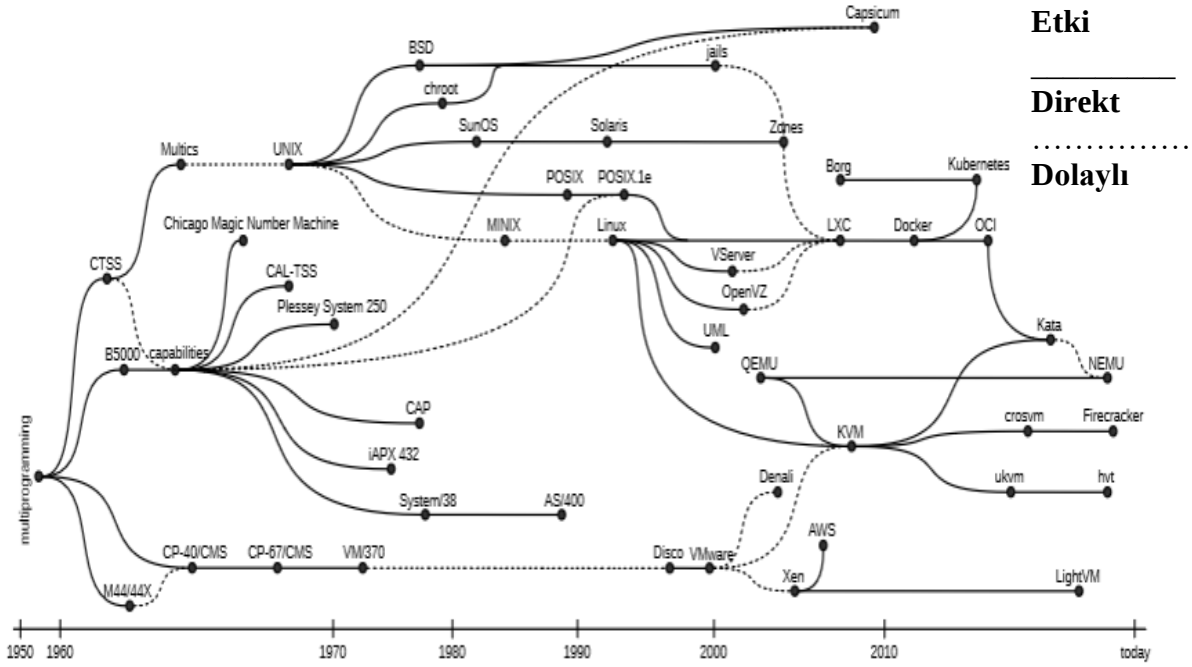
2001 yılında, Linux VServer, kaynak kullanım limit özellikleri ekleyerek, dosya sistemleri, ağ adresleri ve bellek için yalıtımın sağlanması amacıyla Linux çekirdeğine bir dizi yama uygulamıştır. Daha sonra OpenVZ olarak piyasaya sürülen Virtuozzo da dosya sistemleri, prosesler, kullanıcılar, cihazlar ve prosesler arası iletişim (IPC) için kaynak kullanımı ve yalıtımı sağlamak üzere Linux Kernel'e yama uygulamıştır [19].

2002'de, Linux Çekirdeği (2.4.19 sürümü) dosya sistemi ad alanları özelliğini tanıtmıştır. 2006'da Biederman tarafından Linux çekirdeğindeki ad alanı yalıtımı fikrinin dosya sisteminin ötesine proses kimliklerine, IPC'ye, ağ yığınına ve kullanıcı kimliklerine kadar genişletilmesi önerisi çekirdek geliştiricileri tarafından kabul görmüş ve çekirdeğin 2.6.19-3.8 sürümlerine bu özellikler uygulanmıştır [19].

Çekirdeğe uygulanan son yama setini; ayrıcalıksız bir kullanıcının ad alanı oluşturmasına ve bu ad alanı içindeki prosesler için başka bir prosese tam ayrıcalıklar vermesine izin veren, ayrıca bu ad alanı dışındaki prosesler içinse hiçbir ayrıcalık tanımayan kullanıcı ad alanları oluşturmuştur [19].

2006 ile 2007 yılları arasında Rohit Seth ve Paul Menage daha sonradan Cgroups (control groups) olarak bilinen proses kapsayıcıları ile prosesler için kaynak sınırlaması ve proses önceliklendirmesi sağlayan bir yama üzerinde çalışmışlardır [19].

Modern kapsayıcılar gücünü tek bir özellikten değil, kaynak kontrolü ve yalıtım için birden fazla özelliğin kombinasyonundan almaktadırlar. 2008 yılında, Linux kapsayıcılar (LXC), Linux çekirdeğinin Cgroup, ad alanları ve yeteneklerini kullanarak düşük seviyeli sistem kapsayıcıları yaratmak ve başlatmak için oluşturulmuştur. Sanal makinelerin ve kapsayıcıların gelişimine ilişkin evrimsel süreç Şekil 3.2 ile gösterilmiştir.



Şekil 3.2 : Sanal makinelerin ve kapsayıcıların zamanla geçirdiği evrim [19]

3.2.2 Kapsayıcı Tabanlı Sanallaştırma Mimarisi

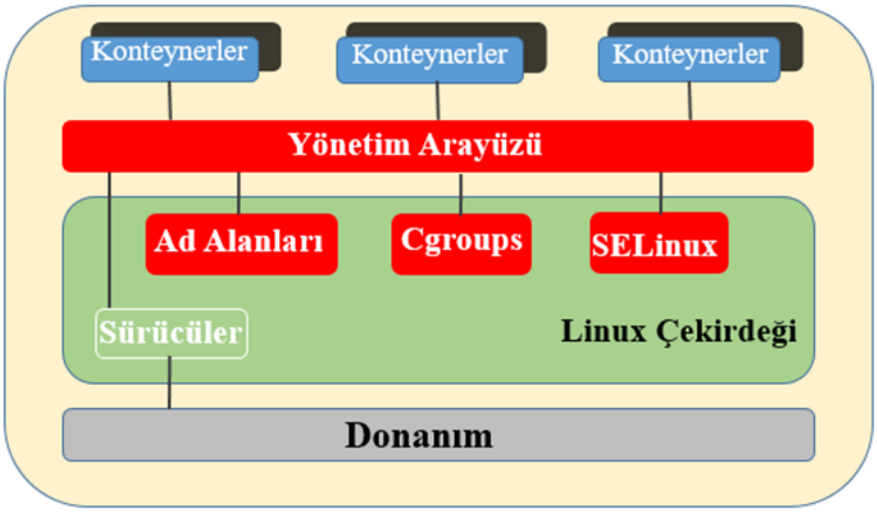
Kapsayıcılar, bulundukları ana makineye ait işletim sistemi üzerinde çalışan, ana makineye ait çekirdek üzerinde bağımsız süreçlere ve ad alanına sahip yapılardır. Kapsayıcıların, hipervizör tabanlı sanal sunuculardan farklı olarak, kendilerine ait bir işletim sistemleri bulunmamaktadır. Konak işletim sistemi üzerinde çalıştırılan bir veya birden fazla kapsayıcının her birine ait yeni bir proses ve yalıtılmış süreçler bulunmaktadır. Kapsayıcı tabanlı bir sistem; bir kök dosya sistemi, güvenli şekilde paylaşılan bir dizi sistem kitaplığı ve yürütülebilir dosyadan oluşan paylaşılabilen sanallaştırılmış bir işletim sistemi sunar. Her kapsayıcı, normal bir işletim sistemi gibi önyüklenebilir, kapatılabilir ve yeniden başlatılabilir. Kapsayıcıların yönetimi için birçok araç bulunmakla beraber Linux kapsayıcı (LXC) ve Docker en yaygın olarak kullanılan yönetim araçlarından olup bu çalışma kapsamında kapsayıcı tabanlı sanallaştırma mimarisinin anlatımı için LXC tercih edilmiştir.

Linux kapsayıcılar ile prosesleri diğer proseslerden ayırmak için bir araç seti oluşturmak amaçlanmıştır. LXC, bir proses ve onun alt prosesleri için kaçışın mümkün olmadığı kapsayıcılar oluşturmaya, saldırgan kapsayıcıdan kaçmayı başarsa bile sisteme erişimini engellemeye çalışmaktadır. Ayrıca LXC, kapsayıcıların kaynak tüketimini sınırlandırmak için bir arayüz sağlar. Kapsayıcılar, ağ birimlerini tamamen sanallaştırarak çekirdek arabirimlerine yalnızca güvenli yollarla erişim yapılmasını sağlar.

Çekirdek ad alanları; prosesleri yalıtmaya, prosesler arası iletişimi veya çekirdeğin ağ alt sistemi gibi alt sistemleri oluşturmaya izin veren bir özelliktir. Çekirdek ad alanları; yalıtılması gereken prosesler için farklı ad alanları yaratarak birbirinden ayrıştırılan esnek bir mekanizma oluşturur. Çekirdek, ağ aygıtları veya işlem / kullanıcı / grup kimlikleri gibi kaynakların ad alanlarına aktarılmasına izin verir ve bu kaynakların senkronizasyonunu yönetir.

Kontrol grupları (Cgroups) prosesleri diğer proseslerden ayırmak için kullanılan, zorunlu olmayan bir özelliktir. Cgroups, alt prosesler de dahil olmak üzere prosesleri ve proses gruplarını izlemek için bir mekanizma sağlamaktadır. Cgroups özelliği proses yalıtımı ile ilgili veya proses yalıtımını daha güçlü hale getirmeye yönelik problemleri çözmezler, ancak yaptıkları hassas kaynak kontrolü ve kaynak sınırlaması ile ana sistemde çalışan kapsayıcı güvenliğine katkı sağlarlar.

Linux'te ve Unix'te, her şey bir dosyadır ve her dosya, hangi kullanıcıya ve hangi gruba ait olduğu bilgisi ile dosya sahibinin, sahip grubun ve sistemdeki diğer herkesin bu dosya ile ne yapabileceği hakkında bilgiler içerir. Bir dosya üzerinde okuma, yazma veya çalıştırma yetkisinin hangi kullanıcı ve gruplar tarafından yapılacağına ilişkin karar isteğe bağlı erişim kontrolü (DAC) vasıtasıyla yapılmaktadır. İsteğe bağlı erişim kontrolünün aksine zorunlu erişim kontrolü (MAC) bu özelliklere bakmaz. Zorunlu erişim kontrolü ile bir kaynağa erişim izni istendiğinde, yetkilendirme politikaları gözden geçirilir ve politika tarafından tanımlanan gereksinimler karşılanıyor ise gerekli izinler verilir. Bu sistemin uygulanmasında SELinux, AppArmor ve grsecurity'e ait RBAC olmak üzere üç farklı yaklaşım bulunmaktadır. Zorunlu erişim kontrolü politikaları ile genellikle hassas olan kaynaklara erişimi kısıtlamak için kullanılır. Sistemde saldırganın, kök kullanıcı olduğu senaryolarda dahi zorunlu erişim kontrolü politikalarında oluşturulmuş ve erişim kısıtı bulunan birtakım işlemleri gerçekleştirmesi mümkün değildir [21]. Şekil 3.3 'te görüldüğü gibi Linux kapsayıcıların çalışması için tüm bileşenler Linux çekirdeği vasıtasıyla sağlanmaktadır.



Şekil 3.3 : Linux kapsayıcı mimarisi [25]

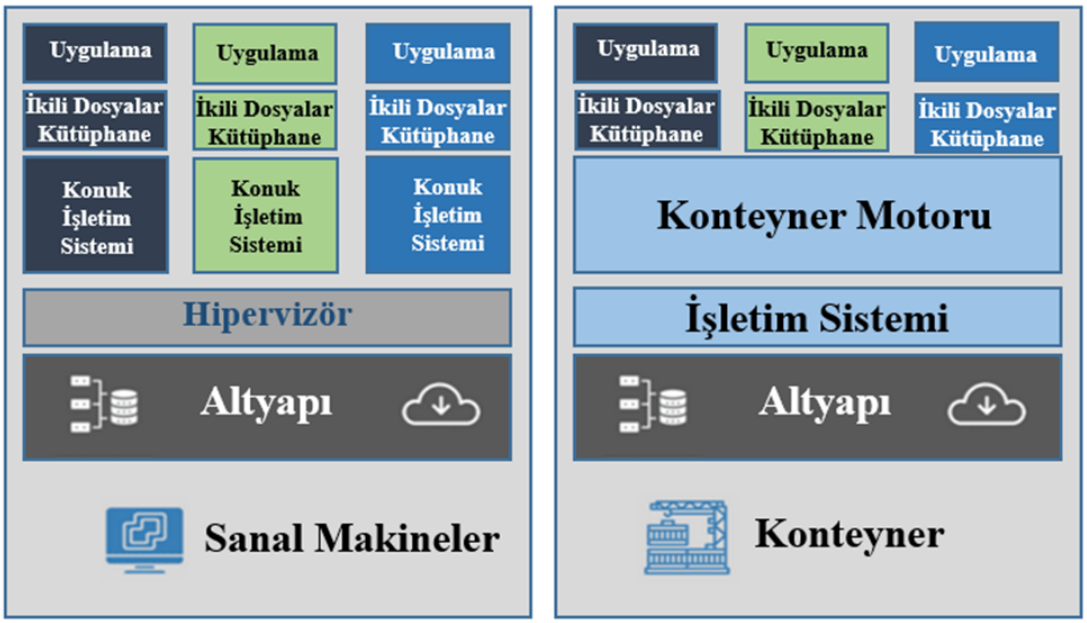
3.3 Sanallaştırma Teknolojilerinin Karşılaştırılması

Mimarideki farklılıklar, kapsayıcı tabanlı sanallaştırmaya hipervizör tabanlı sanallaştırmaya kıyasla bazı faydalar sağlamaktadır. Kapsayıcılar vasıtasıyla gerçekleştirilen sanallaştırmayla, birden fazla uygulamanın aynı donanm üzerinde çalıştırılması donanım maliyetlerini ciddi oranda düşürmekte ve donanım

kaynaklarının kullanımında oluşacak kaynak israfını engellemektedir. Kapsayıcılar tüm bir işletim sistemini içermediğinden, bir uygulamayı çalıştırmak için gerekli boyut ve kaynaklar, aynı uygulamayı çalıştıran bir sanal makinenin ihtiyaç duyduğu kaynaklardan çok daha azdır. Dolayısıyla aynı fiziksel makine üzerinde çalıştırılabilecek kapsayıcı sayısı sanal makine sayısına göre fazladır. Sanal makineden farklı olarak, kapsayıcıların işletim sistemi çekirdeğini önyüklemesi gerekmez, bu sebeple kapsayıcılar bir saniyeden daha kısa bir sürede oluşturulabilir. Düzgün yapılandırılmış kapsayıcılar, tek bir sunucuda çalışan birden çok uygulamadan daha güvenli, ancak KVM sanal makinelerinden ise daha az güvenli bir profile sahiptir [26].

Kapsayıcılar bir uygulamadan daha büyük dosyaları ve çalıştırılması gereken tüm dosyaları paketlemek yerine genellikle mikro hizmet olarak bilinen belirli görevleri gerçekleştiren fonksiyonları paketlemek için kullanılırlar. Dolayısıyla kapsayıcıların bu hafif yapısı megabaytlarla ölçeklenmelerine olanak sağlamaktadır. Ayrıca kapsayıcıların bu hafif yapısı onların birden çok ortamda taşınmalarına yardımcı olmaktadır [27].

Kapsayıcı tabanlı sanallaştırma, ana makineye çok az ek yük getirdiği veya hiç ek yük getirmediği için, kapsayıcı tabanlı sanallaştırmayla neredeyse fiziksel makine performansına yakın bir performans elde edilir. Sanal makineler ile kapsayıcılar arasındaki mimari farklılıklar Şekil 3.4 ile gösterilmiştir. Kapsayıcılar ile sanal makineler arasında yapılan faktörlerin karşılaştırması ise Çizelge 3.1’ de gösterilmiştir.



Şekil 3.4 : Sanal makine ile kapsayıcılar arasındaki mimari farklar [28]

Çizelge 3.1 : Kapsayıcılar ve sanal makinelerin karşılaştırılması

Karşılaştırma Faktörleri	**Kapsayıcılar**	**Sanal Makineler**
İşletim Sistemi Desteği	Ana makine işletim sistemi paylaşılmaktadır.	Her bir sanal makinenin kendisine ait işletim sistemi bulunmaktadır.
Başlatma Zamanı	Çok kısa bir zaman dilimi içerisinde başlatılabilir.	Normal işletim sistemi başlatmak için gerekli olan zamana ihtiyaç duyar.
Taşınabilirlik	Taşınabilirlik kolaydır.	Taşınabilirlik kapsayıcılara kıyasla daha zordur.
Kaynak ihtiyacı	Kaynak ihtiyacı azdır.	Kaynak ihtiyacı yüksektir.
Güvenlik	Çekirdek kapsayıcılar arasında paylaşıldığı için güvenlik problemleri oluşabilir.	Sanal makinede çalışan ayrı işletim sistemi kullanıcı verilerine yüksek güvenlik sağlamaktadır. Güvenlik hipervizör katmanı ile ilişkilidir.
Yedeklilik	Tek bir işletim sistemi bulunduğundan verinin yedekliliği düşüktür.	Her sanal makinenin kendi işletim sistemi olduğu için veri yedekliliği fazladır.
Donanıma Erişim	Donanıma doğrudan erişim bulunmaktadır.	Donanıma doğrudan erişim bulunmamaktadır.
Bellek İhtiyacı	Ana makineye ait işletim sistemi kapsayıcılar arasında paylaşıldığı için daha az bellek miktarı gerekmektedir.	Her bir sanal makinenin kendi işletim sistemi olduğu için büyük bellek miktarına ihtiyaç duymaktadır.
Dosya sistemi ve kitaplık dosyalarının Paylaşımı	Dosya ve kitaplık dosyaları paylaşılabilir.	Dosya sistemi ve kitaplık dosyalarının paylaşılması mümkün değildir.

4. DOCKER KAPSAYICI SANALLAŞTIRMA PLATFORMU

Docker, kapsayıcı adı verilen paketlerin işletim sistemi düzeyinde sanallaştırmasını yaparak barındırdıkları uygulama veya yazılımları geliştirmek, göndermek ve çalıştırmak için oluşturulmuş açık bir platformdur. Kapsayıcılar birbirlerinden kum havuzu olarak ayrılırlar. Kapsayıcılar, yazılımlarını, kitaplıklarını ve yapılandırma dosyalarını tek bir paket haline getirerek çalıştırır, birbirleriyle iyi tanımlanmış kanallar vasıtasıyla iletişim kurarlar.

Mart 2013'te Docker, Docker'ın kurucusu Solomon Hykes'ın tarafından PyCon'da gerçekleştirmiş olduğu konuşması sırasında ilk kez tanıtılmıştır. Docker başladığında, varsayılan yürütme ortamı olarak LXC'yi kullanmaktaydı. Yürütme ortamı olarak LXC'yi kullanması uzun sürmemiş ve yaklaşık bir yıl sonra LXC, Go programlama dilinde yazılmış bir yürütme ortamı olan libcontainer ile değiştirilmiştir [28]. Libcontainer'a geçiş, Docker'ın ad alanlarını, Cgroup'larını, AppArmor profillerini, ağ arayüzlerini ve güvenlik duvarı kurallarını, LXC gibi harici bir pakete bağlı kalmadan, kontrollü ve öngörülebilir bir şekilde serbestçe işlemesine olanak sağlamıştır.

2013 yılında Red Hat, Fedora projesi kapsamında Docker ile işbirliği yapmaya başlamıştır. 2014 yılının sonlarında Microsoft Docker'ı Windows sunucusuna entegre etmiş ve Windows üzerinde yerel destek sağlamaya başlamış olup, aynı yıldan itibaren Docker, Amazon EC2 ve IBM bulut hizmetleri tarafından da kullanılmıştır [28].

Piyasaya sürüldüğünden bu yana, hem açık kaynak topluluğundan hem de ticari yazılım şirketlerinden birçok kullanıcının ilgi odağı haline gelen Docker, kapsayıcı pazarının %27.93'sine sahiptir [29].

4.1 Docker Kapsayıcı Mimarisi

Docker, bir uygulamayı ve tüm bağımlılıklarını, herhangi bir Linux sunucusunda çalıştırılabilen sanallaştırılmış bir kapsayıcı içerisinde paketleyebilir. Bu tür kapsayıcılar sayesinde uygulamanın sürekli entegrasyon (CI) sunucusu, genel bulut veya özel bir bulut gibi herhangi bir ortamdan bağımsız olarak, ilave ayarlara gerek

duymadan oluşturulan uygulamaların çeşitli ortamlarda yürütülmesi için gerekli esnekliği ve taşınabilirliği sağlamaktadır. Docker, kapsayıcıların tek bir Linux içinde çalışmasına izin vermek için Linux çekirdeği tarafından sağlanan Cgroups ve çekirdek ad alanları gibi kaynak yalıtım özelliklerinden ve OverlayFS gibi birleşim yeteneğine sahip bir dosya sisteminden faydalanmaktadır. Docker kapsayıcıların hafif yapısı sayesinde, tek bir sunucu veya tek bir sanal makine üzerinde aynı anda birden fazla kapsayıcı çalıştırılabilmektedir. Yapılan araştırmalarda, tipik bir Docker kullanım senaryosunun ana bilgisayar başına 8 kapsayıcı içerebileceğini ve analiz edilen kuruluşların yaklaşık %25'inin ana bilgisayar başına 18 veya daha fazla kapsayıcı kullandıkları tespit edilmiştir [30].

Linux çekirdeğinin ad alanları için sağladığı destek vasıtasıyla, uygulamanın, işlem ağaçları, ağ, kullanıcı kimlikleri ve takılı dosya sistemleri dahil olmak üzere işletim ortamına yönelik pek çok özellik yalıtılabilir. Çekirdeğin cgrupları, kapsayıcılar tarafından kullanılan kaynakların kontrol altında tutulabilmesi maksadıyla bellek ve CPU için kaynak sınırı sağlayabilir. Docker, sürüm 0.9'dan bu yana libvirt ve LXC aracılığıyla soyutlanmış sanallaştırma arabirimlerini kullanmasının yanında, Linux çekirdeği tarafından sağlanan sanallaştırma yeteneklerini de kullanmak için libcontainer adını verdiği bileşeni oluşturmuştur.

Docker yazılımı; dockerd adı verilen Docker daemon, Docker nesneleri ve Docker kayıt defterleri olmak üzere üç ana bileşenden oluşmaktadır.

4.1.1 Docker Daemon

Docker daemon, çalışan tüm Docker kapsayıcılarını yöneten ve tüm kapsayıcı nesnelerini işleyen kalıcı bir prosestir. Dockerd, Docker motoru API aracılığı ile gönderilen tüm istekleri izlemektedir. Docker istemcisi, kullanıcıların Docker motoru API'lerini uygun argümanlarla çağırarak, Docker daemonlarıyla etkileşime girmesine olanak tanıyan bir komut satırı arabirimi sağlar.

Bir kapsayıcı başlatmak için “docker run” komutu kullanıldığında, Docker istemcisi bu komutu bir HTTP API çağrısına çevirir ve Docker daemon'a gönderir. Docker daemon daha sonra isteği değerlendirir, ana makineye ait işletim sistemiyle görüşür ve kapsayıcının çalışmasını sağlar. Docker daemon, istemcinin istediği şekilde kapsayıcı görüntülerini çeker ve oluşturur.

4.1.2 Docker Nesneleri

Docker nesneleri, Docker'daki bir uygulamanın kodunu ve tüm bağımlılıklarını birleştirmek için kullanılan varlıklardır. Docker nesnelerini kapsayıcılar ve görüntüler oluşturmaktadır.

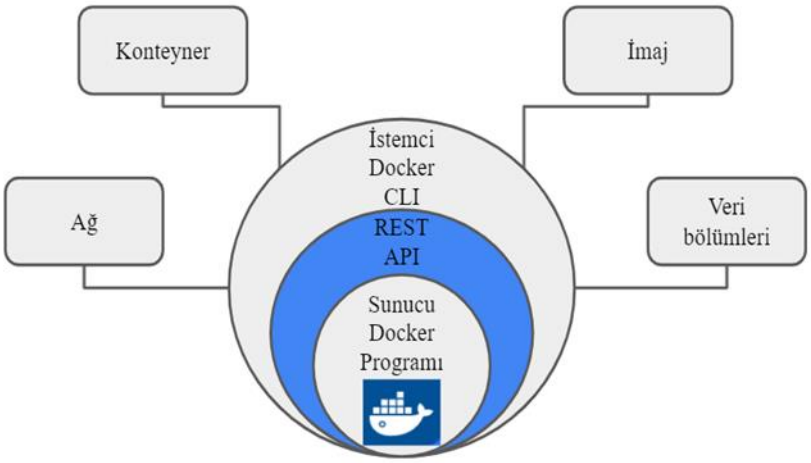
Docker kapsayıcı, Docker görüntülerini kullanarak oluşturulan uygulamaları yalıtılmış bir alanda çalıştıran, standartlaştırılmış ve kapsüllenmiş yapılardır. Bu yapılar içerisinde uygulamaya ait tüm bağımlılıklar ve kütüphaneler bulunmaktadır. Kapsayıcıların çalışma zamanı esnasında yönetimi Docker daemon tarafından yapılmaktadır.

Docker görüntüsü, Docker platformunda çalışabilen bir kapsayıcı oluşturmak için bir dizi talimat içeren salt okunur bir şablon olup, uygulamaları ve tüm bağımlılıkları ile ortam gereksinimlerini depolamak ve göndermek için kullanılır. Kullanıcılar kendi görüntülerini oluşturabilir, Docker Hub veya Docker Cloud adı verilen depolardan ihtiyaç duydukları görüntüleri indirebilirler. Docker görüntülerini oluşturan dosyaların her biri ayrı bir katman olarak bilinmektedir. Bu katmanlar, her katmanın hemen altındaki katmana bağlı olduğu aşamalar halinde üst üste inşa edilen bir dizi ara görüntü oluşturur. Katman hiyerarşisi, Docker görüntülerinin yaşam döngüsü yönetiminde oldukça önemli bir role sahiptir. Görüntü katmanında yapılan herhangi bir değişiklik yalnızca değişikliğin yapıldığı görüntü katmanını değil tüm katmanların yeniden oluşturulmasını gerektirebilir.

4.1.3 Docker Kayıt Defteri

Docker kayıt defteri Docker görüntüleri için bir depolama alanıdır. İnternet üzerinde Docker Hub aracılığıyla sunulan tüm kullanıcıların erişimine açık olan bu depolama alanı; Docker görüntülerine ait versiyon, yayınlanma tarihi, yazar bilgileri gibi bilgiler ve görüntü kullanımına yönelik açıklamalar içermektedir. Kullanıcılar tarafından oluşturulan, özel veya genel depolarda tutulan resmî veya özel görüntüler bu depolara yüklenip indirilebilmektedir [31].

Şekil 4.1' de gösterildiği gibi, Docker aracının; Docker daemon, Docker CLI ve REST API olmak üzere üç ana katmanı bulunmaktadır.



Şekil 4.1 : Docker'a genel bir bakış [32]

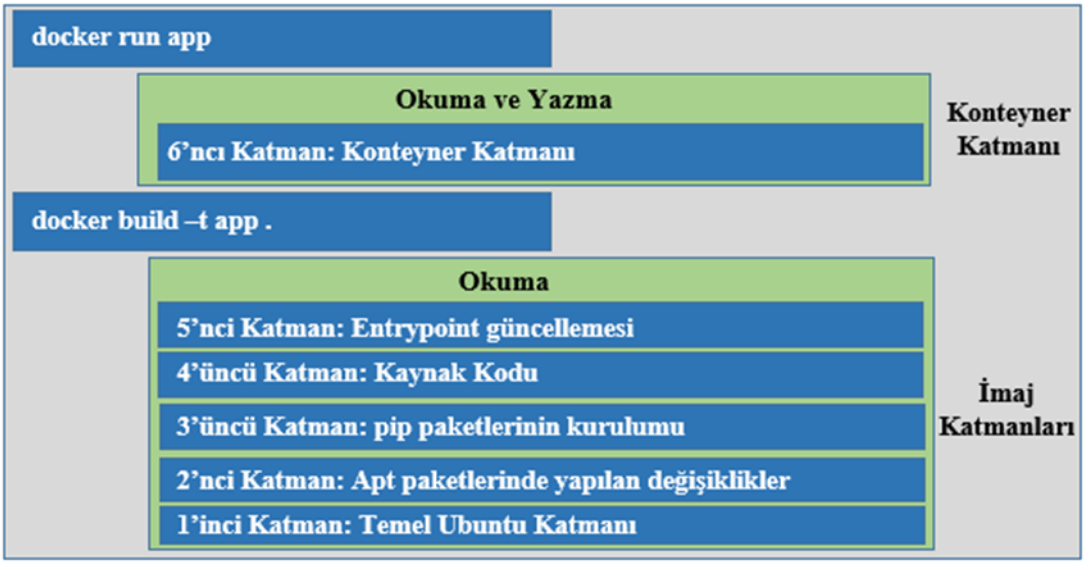
İstemci, Docker CLI vasıtasıyla kullandığı komut satırı betikleri sayesinde Docker ile iletişim kurmaktadır. CLI, komut dosyası oluşturma veya doğrudan CLI komutları aracılığıyla Docker daemon programını denetlemek veya etkileşimde bulunmak için Docker REST API'sini kullanır. Docker daemon programı görüntüler, kapsayıcılar, ağlar ve depolama birimleri gibi Docker nesnelerini oluşturur ve bunları yönetir.

4.2 Docker Dosya Sistemi Yapısı

Docker disk görüntüleri, JavaScript nesne gösterimi (JSON) biçimindeki meta verilerle birlikte bir dizi katmandan oluşmaktadır. Kapsayıcı içinde varsayılan ayarlarda oluşturulan tüm dosyalar, yazılabilir bir kapsayıcı katmanında depolanmaktadır. Katmanlar, kapsayıcının kök dosya sistemi için temel oluşturması amacıyla üst üste istiflenmektedir. Docker depolama sürücüsü, bu katmanları yığınlamaktan ve tek bir birleşik görünüm sağlamaktan sorumludur. Yeni bir kapsayıcı oluşturulduğunda, Docker görüntüsünde bulunan temel katman yığınının üstüne yeni ve ince bir yazılabilir katman eklenmektedir. Kapsayıcıların yazılabilir katmanında bulunan verilerin başka bir yere taşınması zordur. Kapsayıcıda yapılan yeni dosya oluşturma, mevcut dosyaları değiştirme veya dosyaları silme gibi tüm değişiklikler, bu ince yazılabilir kapsayıcı katmanında yapılmaktadır. Her kapsayıcı için ince bir yazılabilir katmanın olması ve tüm değişikliklerin bu kapsayıcı katmanında depolanması, birden çok kapsayıcının aynı temel görüntüyü paylaşabilmelerine ve kendilerine özgü veri durumlarına sahip olmalarına imkan sağlamaktadır. Docker depolama sürücüsü, hem görüntü katmanlarını hem de yazılabilir kapsayıcı katmanını etkinleştirmekten ve yönetmekten sorumludur. Docker görüntü ve kapsayıcı

yönetiminin arkasında istiflenebilir görüntü katmanları ve yazma üzerine kopyalama (copy-on-write) yapabilme yeteneği önemli rol oynamaktadır. Yazma üzerine kopyalama, aynı verilere erişmesi gereken sistem proseslerinin kendi kopyalarını oluşturmak yerine benzer verilerin aynı örneği kullandıkları paylaşım ve kopyalama stratejisidir. Herhangi bir proses verileri değiştirmek veya yazmak isterse, ancak o zaman işletim sistemi bu prosesin kullanması için verilerin bir kopyasını oluşturur. Yalnızca ilgili prosesin oluşturulan veri kopyasına erişimine izin verilir. Diğer tüm prosesler orijinal verileri kullanmaya devam ederler. Docker, hem görüntüler hem de kapsayıcılar için yazma üzerine kopyalama teknolojisini kullanmaktadır. Bu strateji, hem görüntünün disk alanı kullanımının hem de kapsayıcı başlangıç zamanlarının performansının optimize edilmesine katkı sağlar [33].

Görüntüler varsayılan olarak /var/lib/docker/<driver>/ konumunda depolanmaktadır. "<driver>" yolu, gelişmiş çok katmanlı birleştirme dosya sistemi (aufs), B-tree dosya sistemi (Btrfs), sanal dosya sistemi anahtarı (VFS), aygıt eşleyici veya OverlayFS gibi kullanılan depolama sürücülerine göre değişiklik göstermektedir. "docker build" komutu çalıştırıldığında Docker, dockerfile içindeki her komut için bir ayrı bir katman oluşturmaktadır. Bu görüntü katmanları salt okunur katmanlardır. "docker run" komutu çalıştırıldığında Docker, okuma-yazma katmanları bulunan kapsayıcı katmanını oluşturmaktadır.



Şekil 4.2 : Docker kapsayıcı dosya katmanları [33]

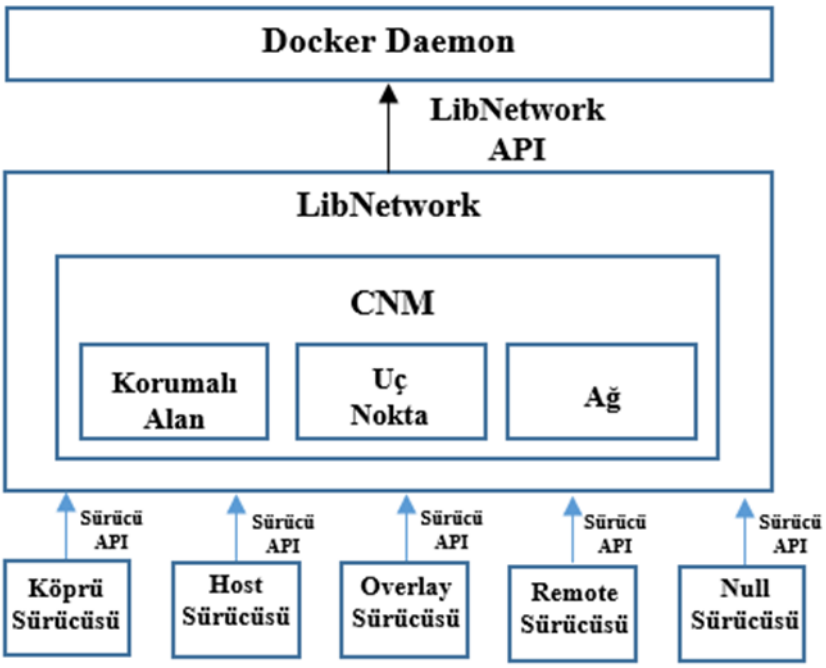
"docker volume" oluşturma komutu kullanılarak /var/lib/docker/volumes dizininde yeni bir Docker birimi oluşturulabilmektedir. Ayrıca verilerin konak işletim sistemi

üzerinde belirli bir konumda tutulmasının gerektiği durumlarda veya konak işletim sistemi depolama alanında mevcut verilerin kullanımına ihtiyaç duyulduğu durumlarda, dosya konumu kullanıcı tarafından kapsayıcı ile ilişkilendirilebilmektedir.

4.3 Docker Ağ Yapısı

Her Docker kapsayıcısının kendisine ait ağ yığını (network stack) bulunmaktadır. Linux çekirdeğinin NET ad alanı özelliği aracılığıyla her bir kapsayıcı için yeni bir NET ad alanı başlatılır. Docker ağı, TCP/IP yığın uygulaması, paket filtreleme ve VXLAN gibi Linux çekirdeği özelliklerinden faydalanır. Mevcut Linux çekirdek özelliklerinin kullanılması, yüksek performans, sağlamlık, farklı dağıtımlar arasında geçiş ve taşınabilirlik gibi çeşitli faydalar sağlar. Docker ağ yapısı, ana makine aracılığıyla Docker kapsayıcılar ile dış dünya arasındaki iletişimini yürütür.

Libnetwork, Docker'ın kapsayıcı ağ modeli (Container Network Management- CNM) ve Linux'e ait libcontainer özelliğinin birleştirilmesiyle oluşturulmuştur. Docker daemon, alt uygulama için LibNetwork tarafından sağlanan API'ler aracılığıyla ağ oluşturma ve ağ yönetimini gerçekleştirmektedir. Kapsayıcı ağ modeli birden fazla ağ sürücüsü desteği ile kapsayıcılar için ağ oluşturma yöntemidir. Kapsayıcı ağ modeli, aynı ağdaki tüm kapsayıcıların birbirleriyle iletişim kurabilmesini, kapsayıcılar ve desteklenen sürücüler arasında çoklu ağ trafiğinin bölümlere ayrılmasını ve kapsayıcıların aynı anda birden fazla ağa dahil olmasını sağlamaktadır. Kapsayıcı ağ modeli; korumalı alan (sandbox), uç nokta (endpoint) ve ağ olmak üzere üç ana bileşen üzerine kuruludur. Korumalı alan, kapsayıcı ağ yığınının yapılandırmasını koruyan, böylece ağ arabirimi, yönlendirme ve DNS yapılandırma yönetimi dahil olmak üzere farklı kapsayıcıların tamamen yalıtılabilmesi için yalıtılmış bir ağ işletim ortamıdır. Korumalı alan, birden çok ağ ve birden çok uç nokta içerebilir. Uç nokta, kapsayıcının ağa eriştiği noktayı temsil etmekte olup fiziksel bir ağ kartı gibi değerlendirilebilir. Kapsayıcı ağ modeli bileşenlerinden olan ağ, Linux köprüsüne veya vlana dayalı olarak birbirleriyle doğrudan iletişim kurabilen bir dizi uç noktayı temsil etmektedir [34]. Kapsayıcı Ağ modeli Şekil 4.3'te gösterilmiştir.



Şekil 4.3 : Kapsayıcı ağ modeli yapısı [34]

Varsayılan ayarlarda Docker platformuna ait bridge, MACVlan, host, overlay ve none olmak üzere beş adet ağ modu bulunmaktadır. Host modu, kapsayıcı ve konak işletim sistemi ağ yığını arasındaki tüm yalıtımı çıkararak, kapsayıcıyı doğrudan konak işletim sistemi ağına ekler. Host modunda kapsayıcı için bir sanal ağ kartı tanımlanmaz ve IP adresi ataması yapılmaz.

None modunda kapsayıcıya herhangi bir ağ arabirimi ataması yapılmaz ve bu mod ile çalıştırılan kapsayıcılar için tam bir ağ yalıtımı sağlanmış olur.

Docker platformu için varsayılan ağ modu bridge'dir. Kapsayıcılar ağa, bridge modunda "docker0" ile temsil edilen bir arayüz vasıtasıyla erişmektedirler. Birden fazla kapsayıcının birbirleriyle ve dışarısıyla iletişimi bridge modunda gerçekleşmektedir. IP adresleri DHCP aracılığıyla otomatik olarak dağıtılmaktadır. Ayrıca kapsayıcılar için bridge modunda özel bir IP ve subnet ataması yapılabilmektedir.

MACVlan modunda kapsayıcı içerisine MAC adresi ataması yapılmaktadır. MAC adres atamasıyla kapsayıcının ağ üzerinde fiziksel bir aygıtmış gibi görünmesi sağlanmaktadır. Docker, MAC adresine göre trafiği kapsayıcılara yönlendirmektedir. MACVlan modu, trafiğin doğrudan fiziksel ağa bağlanması istenen kapsayıcılar için idealdir. MACVlan modu, kapsayıcının konak işletim sistemi ağ yığını üzerinden

yönlendirilmesi yerine doğrudan fiziksel ağa bağlanması nedeniyle, bridge moda kıyasla daha iyi bir performans sergilemektedir.

Overlay ağ sürücüsüyle birden çok konak işletim sistemi üzerinde çalışan Docker kapsayıcılar arasında dağıtılmış bir ağ oluşturulabilir. Overlay ağ modunda, farklı ana makineler arasında kapsayıcıdan kapsayıcıya iletişim sağlamak için fiziksel ağ altyapısı üzerinde üst düzey bir ağ oluşturulmaktadır. Docker kapsayıcılar arasında haberleşmeyi sağlayan bu ağ modelinde VXLAN (Virtual Extensible Lan) teknolojisi kullanılmakta ve diğer katmanlardan yalıtılmış bir haberleşme sağlanmaktadır. VXLAN'lar ile üçüncü katman altyapısının üzerine sanal bir ikinci katman ağı oluşturulmaktadır.

4.4 Docker Hub

Docker Hub, kapsayıcı geliştiricileri, açık kaynak projeleri ve bağımsız yazılım satıcıları (ISV) dahil olmak üzere tüm dünya tarafından kullanılan, sunduğu içerikle dünyanın en büyük Docker kapsayıcı görüntü deposudur. Kullanıcılar tarafından görüntüler depolanmak ve paylaşılmak üzere ücretsiz genel depolara yüklenebilir veya kullanıcıya ait özel depolarda bulundurulabilir. Docker Hub tüm kullanıcılar için merkezi bir buluşma yeri niteliğindedir. Docker Hub aracılığıyla, uygulamaların sürekli entegrasyon ve dağıtım aşamalarında, farklı türdeki müşterilere gerekli destek sağlanmaktadır.

Docker Hub üzerinde Docker görüntüsüne ait kısa bir açıklama ve istemcinin görüntüsü indirmek için kullanacağı komutlar gibi bilgiler de yer almaktadır. MongoDB, nginx, Apache, Ubuntu ve MySQL için resmi görüntüler da dahil olmak üzere bir milyardan fazla kez indirilmiş 100.000'den fazla görüntü barındırmaktadır [15].

4.5 Docker Swarm

Docker Swarm, kümelenmiş (cluster) bir yapılandırma aracılığıyla birbirleriyle etkileşim kurmak üzere ayarlanan, üzerinde Docker kapsayıcılar çalıştıran ve tüm fiziksel makinelerin bir araya gelmesini sağlayan bir orkestrasyon aracıdır [35]. Docker Swarm ana bilgisayar havuzunu tek bir sanal Docker ana bilgisayarına dönüştürmektedir [36]. Docker Swarm yapılandırmasından sonra oluşturulan küme

üzerinde tüm Docker komutları uygulanabilmekte ve çalıştırılması istenen kapsayıcı için en uygun konak işletim sistemine karar veren bir dağıtım stratejisi kullanılmaktadır. Kullanılan bu dağıtım stratejisiyle sistem sürekli çalışır halde bulundurulmakta ve yüksek erişilebilirlik (high availability) sağlanabilmektedir.

Bir Swarm, Swarm modunda çalışan ve yönetici olarak hareket eden birden çok ana makineden oluşmaktadır. Docker Swarm içerisinde yer alan bir ana makine, yönetici, çalışan veya her iki rolü birden gerçekleştirebilmektedir. Yönetici düğümleri, Swarm'ın istenen durumunu korumak için gereken düzenleme ve küme yönetimi işlevlerini gerçekleştirmektedir. Çalışan düğümler ise yönetici düğümlerden gönderilen görevleri alır ve yürütürler. Her çalışan düğümde bir ajan bulunmakta ve çalışan düğümlere atanan görevler hakkında yönetici düğüme rapor vermektedir. Çalışan düğümü atanan görevlerin mevcut durumunu, sistemin sürekli çalışabilir halde bulundurulabilmesi amacıyla yönetici düğüme bildirir.

4.6 Docker Kapsayıcı Güvenliği

Docker platformu ile gerçekleştirilen sanal nesnelerin yalıtımı donanımın sanallaştırılması veya bağımsız bir işletim sisteminin kullanılması esasına dayanmaktadır. Docker çalışan ortamların güvenli yalıtımının sağlanması için Linux'ta ad alanı mekanizmasını kullanmaktadır. Ayrıca bilgisayar kaynaklarının yönetimini gerçekleştirmek için Linux'de Cgroup mekanizması kullanılmaktadır. Güvenliğin güçlendirilmesi için çekirdek yeteneklerinden faydalanılmaktadır.

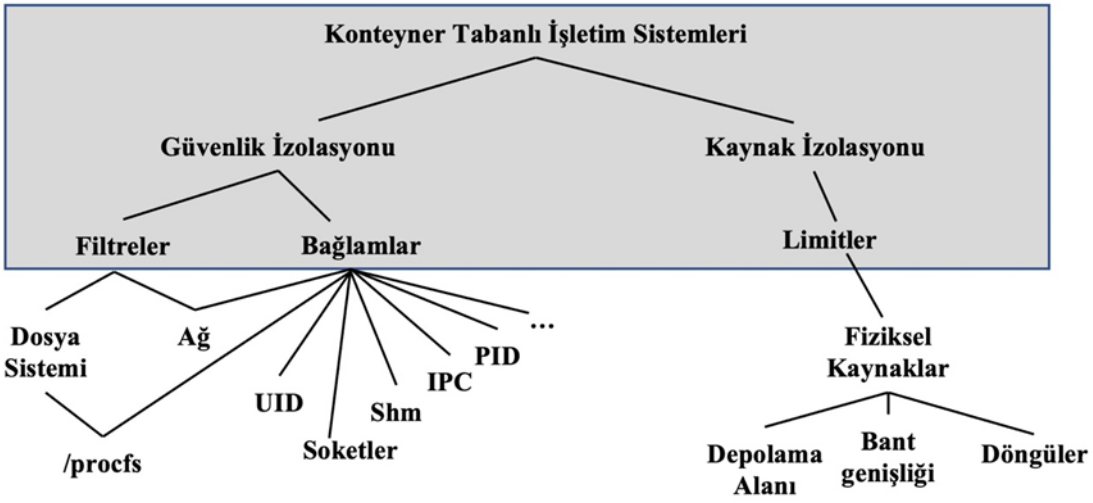
4.6.1 Kaynak, Dosya Sistem ve Cihaz Yalıtımı

Linux'de ad alanı mekanizması aracılığıyla güvenli bir yalıtım sağlanmakta ve bu yalıtım sayesinde diğer kapsayıcıların yalıtılmış kaynaklara erişimi engellenir. Bu mekanizma ile bilgisayar kaynak erişiminde ihtiyaç duyulan şeffaflık arttırılmış olur. Ad alanıyla her bir kapsayıcı için özel bir ağ yapısı, dosya sistemi ve kendi iç proseslerinin iletişim sağlayabildiği bir yapı oluşturulmaktadır.

Proses yalıtımının temel amacı, güvenliği ihlal edilmiş kapsayıcıların diğer kapsayıcılara müdahale etmesini engellemek için proses yönetim arabirimlerinin kullanılmasının önlenmesidir. Docker, kapsayıcılarda çalışan prosesleri ad alanlarına sarmaktadır. Kapsayıcının izinleri ve görünürlüğü diğer kapsayıcı ve konak işletim sistemi üzerinde çalışan proseslerle sınırlanarak proseslerin yalıtımı

gerçekleştirilmektedir. Bu mekanizma, kapsayıcının proses kimlik numarası alanını konak işletim sisteminden ayıran PID ad alanlarının desteklenmesiyle çalışmaktadır. PID ad alanlarının hiyerarşik olması nedeniyle, bir proses diğer prosesleri yalnızca kendi ad alanında veya "alt" ad alanlarında görebilmektedir.

Sonuç olarak, yeni bir ad alanı oluşturulduğunda ve kapsayıcıya atandığında, konak işletim sistemi kapsayıcının yeni PID ad alanı içindeki prosesleri izleyebilir ve etkileyebilir, ancak kapsayıcı içerisindeki prosesler, konak işletim sistemi üzerinde veya diğer kapsayıcılara ait prosesler üzerinde herhangi bir değişiklik yapamamaktadır. Ayrıca PID ad alanları her bir kapsayıcının kendi "init" benzeri prosesine (PID 1) sahip olmasına izin verir. Bu sebeple, ad alanındaki tüm proseslerin sonlandırılması kapsayıcının sonlandırılması anlamına gelmektedir. Bu yöntem sayesinde yönetici şüpheli bir işlem ile karşılaştığında kolaylıkla kapsayıcıyı kapatabilmektedir.



Şekil 4.4 : Docker kapsayıcı yalıtımı [37]

Dosya sistemi yalıtımının sağlanması için, konak işletim sistemi ve kapsayıcı dosya sistemlerinin, yetkisiz erişim ve değişikliklerden korunması gerekmektedir. Docker, farklı kapsayıcılarla ilişkili dosya sistemi hiyerarşisinin yalıtımı için dosya sistemi ad alanları olarak da adlandırılan bağlama (mount) ad alanlarını kullanmaktadır. Bağlama ad alanları, her bir kapsayıcının işlemlerine dosya sistemi ağacının farklı bir görünümünü sağlayarak kapsayıcıda meydana gelen tüm bağlama olaylarını yalnızca kapsayıcı içinde etki yaratacak şekilde kısıtlamaktadır. /sys, /proc/sys, /proc/sysrq-trigger, /proc/irq ve /proc/bus gibi bazı çekirdek dosya sistemlerinde ad alanı bulunmamakta ve bir Docker kapsayıcının çalışması için kapsayıcının bu dosyaları

kullanabilmesi gerekmektedir. Bu durum dosya sistemlerinin görünümünü konak işletim sisteminden devralması ve bunlara doğrudan erişebilmesi sorununa neden olmaktadır.

Docker, güvenliği ihlal edilmiş bir kapsayıcının dosya sistemleri aracılığıyla konak işletim sistemine yapabileceği tehditleri, kapsayıcılardan bu dosya sistemlerine yazma iznini kaldırarak ve herhangi bir kapsayıcı işleminin herhangi bir dosya sistemini yeniden bağlamasına izin vermeyerek engellemektedir. Bir diğer kontrol mekanizması ise kapsayıcılardan Linux sistemlerde kök kullanıcı yetkilerine erişim sağlayabilen CAP_SYS_ADMIN yeteneğinin kaldırılmasıdır. Aynı görüntü üzerinde birden çok kapsayıcı oluşturulması durumunda, her bir kapsayıcı içeriği kendi özel dosya sistemine yazılmaktadır.

Unix sistemlerde çekirdek ve uygulamalar, donanıma ve aygıt sürücülerine arabirim görevi gören özel dosyalar olan aygıt düğümleri aracılığıyla erişmektedir. Kapsayıcılar, /dev/mem (fiziksel bellek), /dev/sd∗ (depolama) veya /dev/tty (terminal) gibi bazı önemli aygıt düğümlerine erişebiliyorlarsa, konak işletim sistemine ciddi zarar verebilirler. Bu nedenle, bir kapsayıcının erişebileceği aygıt düğümleri kümesini sınırlamak oldukça önemlidir. Cgroups'un Aygıt Beyaz Listesi Kontroller özelliği, Docker'ın bir kapsayıcının erişmesine izin verdiği aygıt kümesini sınırlamak için gerekli araçları sağlamaktadır.

Docker varsayılan ayarlarda kapsayıcılara genişletilmiş ayrıcalıklar vermemektedir. Bu nedenle, varsayılan ayarlarda çalıştırılan kapsayıcılar herhangi bir cihaza erişemezler. Ancak kullanıcının bir kapsayıcıyı "ayrıcalıklı" olarak çalıştırması durumunda, Docker kapsayıcıya tüm cihazlara erişim izni verilmiş olacaktır.

4.6.2 Prosesler Arası İletişim

IPC (işlemler arası iletişim); semaforlar, mesaj kuyrukları ve paylaşılan bellek bölümleri gibi prosesler arasında veri alışverişini sağlayan bir işletim sistemi mekanizmasıdır. Kapsayıcılarda çalışan proseslerin, yalnızca belirli bir dizi IPC kaynağı aracılığıyla iletişim kurabilmeleri ve diğer kapsayıcılar ile konak işletim sistemine ait proseslere müdahalelerine izin verilmemesi için kısıtlanması gerekmektedir. Docker, ayrılmış IPC yalıtımının sağlanması için her bir kapsayıcının IPC ad alanlarını oluşturmasına izin veren IPC ad alanı mekanizmasını

kullanmaktadır. IPC ad alanındaki prosesler, diğer IPC ad alanlarındaki IPC kaynaklarını okuyamaz veya yazamazlar.

4.6.3 Ağ İzolasyonu

Kapsayıcılarda ağ yalıtımı, ortadaki adam (MitM) ve ARP sahtekârlığı gibi ağ tabanlı saldırıların önlenmesi açısından önemlidir. Kapsayıcılar, diğer kapsayıcıların veya konak işletim sisteminin ağ trafiğini gizlice dinleyemeyecek veya değiştiremeyecek şekilde yapılandırılmalıdır. Docker her kapsayıcı için ağ ad alanlarının kullanarak bağımsız bir ağ yapısı oluşturmaktadır. Bu nedenle, her kapsayıcının kendi IP adresleri, IP yönlendirme tabloları ve sanal ağ kartları bulunmaktadır. Kapsayıcıların varsayılan ayarlarda ana makineye olan bağlantıları sanal ethernet köprüsü kullanılarak sağlanmaktadır. Bu yaklaşımda Docker, ana makinede paketleri ağ arabirimleri arasında otomatik olarak ileten docker0 adlı sanal bir ethernet köprüsü kullanmaktadır.

4.6.4 Kaynakların Kontrolü

Hizmet dışı bırakma (DoS), bir işlemin veya bir grup işlemin sistemin tüm kaynaklarını tüketmeye çalıştığı ve böylece diğer işlemlerin normal işleyişini bozduğu, çok kiracılı sistemlerde görülen yaygın saldırılardan biridir. Bu tür saldırıların önlenmesi için her bir kapsayıcıya tahsis edilen kaynakların sınırlandırılması gerekmektedir. Herhangi bir Docker kapsayıcının kullanabileceği CPU, bellek ve disk G/Ç gibi kaynaklar kontrol edilerek, kapsayıcıların kaynakları adil bir şekilde kullanması sağlanarak, herhangi bir kapsayıcının tüm kaynakları tüketmesi engellenmektedir. Ayrıca Docker, her bir kapsayıcıya tahsis edilen kaynaklarla ilgili sınırların ve kısıtlamaların yapılandırılmasına olanak sağlar.

4.6.5 Çekirdek (Kernel) Limitleri

Linux ana bilgisayar sisteminin güvenliğini güçlendirmek için Linux yetenekleri ve Linux Güvenlik Modülü (LSM) de dâhil olmak üzere bazı çekirdek güvenlik sistemleri mevcuttur. Linux yetenekleri sayesinde her işleme atanan ayrıcalıklar kısıtlanabilmektedir. LSM, Linux çekirdeğinin farklı güvenlik modellerini desteklemesine izin veren bir çerçeve sağlamaktadır. Resmî Linux çekirdeğine entegre edilen LSM'ler arasında AppArmor, SELinux ve Seccomp bulunmaktadır.

Unix sistemler; prosesleri ayrıcalıklı prosesler (süper kullanıcı ve kök kullanıcı) ve ayrıcalıksız prosesler olmak üzere iki kategoride sınıflandırmaktadır. Docker kapsayıcılar, konak işletim sistemiyle paylaşılan bir çekirdek üzerinde çalışmaktadır. Dolayısıyla görevlerinin çoğu ana bilgisayar tarafından gerçekleştirilmektedir. Bir kapsayıcıya tam kök ayrıcalıkları sağlamak gereksizdir. Bu nedenle, bir kapsayıcıdan bazı kök yeteneklerinin kaldırılması, kapsayıcının kullanılabilirliğini veya işlevselliğini etkilemez, ancak sistemin güvenliğini önemli bir şekilde artırır. Saldırganın varsayılan ayarlar ile çalıştırılan bir kapsayıcı içinde kök erişimi elde etmiş olması durumunda bile konak işletim sistemine zarar vermesini engelleyen Linux güvenlik özellikleri bulunmaktadır.

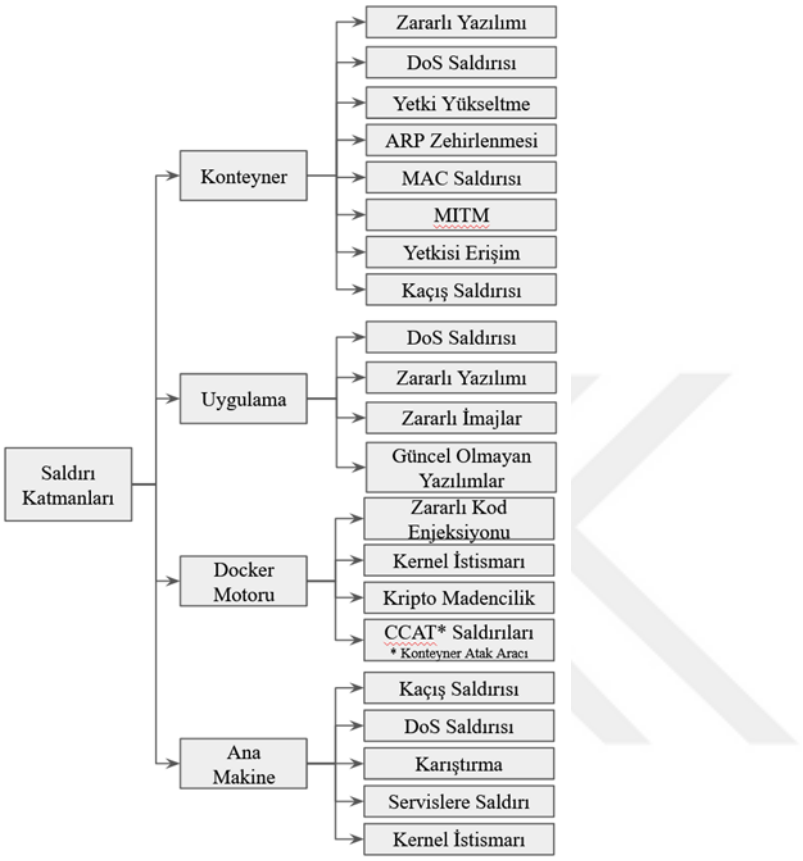
SELinux, Linux sistemine yönelik bir güvenlik geliştirmesidir. Linux, bir nesneye erişimi kontrol etmek için nesnenin sahibi/grubu ve izin bayrakları gibi standart DAC (Discretionary Access Controls) mekanizması ile birlikte gelmektedir. SELinux'te her şey etiketler tarafından kontrol edilmektedir. Her dosya/dizin, proses ve sistem nesnesinin bir etiketi vardır. Sistem yöneticisi, prosesler ve sistem nesneleri arasındaki erişimi kontrol etmek ve kurallar yazmak için bu etiketleri kullanır. SELinux, bir kapsayıcı içindeki işlemlerin, kapsayıcıların dışındaki nesnelere kök ayrıcalıklarıyla bile uygun olmayan şekilde erişimini engellemekte ve kapsayıcılar için güvenli bir alan sağlamaktadır.

AppArmor, SELinux gibi kapsamını bireysel programlarla sınırlayan zorunlu erişim kontrolüne dayalı, Linux güvenlik mekanizmasıdır. Kullanıcıya, her program için, programın yeteneklerini sınırlayan bir güvenlik profili yüklemesine izin vermektedir. Docker, varsayılan ayarlarda AppArmor destekli sistemlerde, yeni bir kapsayıcı başlatırken önceden tanımlanmış, /sys/fs/cgroups/ ve /sys/kernel/security/ gibi konak işletim sisteminde önemli dosya sistemlerine erişimi reddeden bir AppArmor profili kullanmaktadır.

4.7 Docker Kapsayıcı Güvenlik Tehditleri

Tomar ve diğerleri tarafından yapılan çalışmada kapsayıcılara yönelik saldırılar, kapsayıcı, uygulama ve ana makine olmak üzere üç farklı katmanda incelenmiştir. Kapsayıcılara yönelik saldırılar; zararlı yazılım, kapsayıcı kaçış saldırısı, ARP kandırmaca (ARP spoofing) ve MAC adres taşması (MAC Flooding) saldırıları, ortadaki adam saldırısı (Man-in-the-Middle), yetkisiz erişim, zehirli görüntüler

(poisoned images), eski yazılım, bulut kapsayıcı saldırı aracı (CCAT), zararlı kod enjeksiyonu, çekirdek istismarı (Kernel exploit), kripto madencilik zararlısı (cryptojacking), tampering ve gereksiz servislere yönelik saldırılar olarak Şekil 4.5'te sınıflandırılmıştır.



Şekil 4.5 : Saldırı taksonomisi [14]

Zararlı Yazılım ile yapılan saldırılar; kurbanın saldırgan tarafından kontrol edilen bir web sayfasına yönlendirilip, kötücül yazılım barındıran bir dosyayı çalıştırmasını veya daha önceden hazırlanmış görüntü içerisine kötücül yazılım barındıran dosya veya kod betiklerinin çalıştırılmasını sağlayarak gerçekleştirilmektedir.

Kapsayıcılara yönelik yapılan hizmet dışı bırakma saldırıları, kapsayıcıya ait kaynakların tüketilmesiyle veya kapsayıcı içerisinde çalışan uygulamaların çökmesine neden olacak şekilde çok sayıda istek gönderilmesiyle yapılmaktadır. Kaynak tüketimini hedef alan saldırılarda, saldırgan işlemci ve bellek gibi kaynakların yetkili prosesler tarafından kullanımının kısıtlanmasını ya da engellenmesini amaçlar. Kapsayıcı tabanlı mimaride tüm kapsayıcıların konak işletim sistemine ait çekirdek kaynağını paylaşıyor olmaları, bu tür saldırılarda diğer kapsayıcıları da etkileyerek sağlıklı hizmet vermelerini engelleyebilir.

Yetki yükseltme saldırılarında, saldırgan çekirdeğe ait kök ayrıcalığını kazanmaya çalışmaktadır. Saldırgan tarafından gerçekleştirilen bellek modifikasyon saldırıları ile bellekte bulunan belirli veri yapılarının değiştirilmesi kontrol akışını bozmakta ve saldırgana sistemde kök kullanıcı hakları vermektedir. Saldırgan tarafından yapılan dosya modifikasyonları, süper kullanıcı parolasını değiştirmek veya kötü amaçlı programları kök kullanıcı ayrıcalıklarıyla çalıştırmak için kullanılmaktadır.

Kapsayıcılar ile ana makine arasında ağ erişimi için sanal ethernet köprüsü (virtual ethernet bridge) kullanılmaktadır. Docker tüm ağ paketlerini bu köprü aracılığıyla iletmektedir. Docker tarafından yeni bir kapsayıcı oluşturulduğunda yeni bir sanal ethernet arabirimi kurulur ve bu köprüye bağlanır. Docker'ın varsayılan bağlantı modeli, gelen paketler üzerinde herhangi bir filtreleme yapmadan köprü tarafından istenen arayüzlere iletildiğinden, ARP zehirlenmesi ve MAC saldırılarına karşı savunmasızdır. Saldırgan, gönderdiği sahte ARP mesajları ile kendi MAC adresini yetkili bir kullanıcı IP adresi ile ilişkilendirebilir. Ayrıca bu saldırılar ile kötü niyetli bir kullanıcı iki taraf arasında gerçekleştirilen bir iletişime müdahil olabilir ve paylaşılan bilgileri ele geçirip değiştirebilir.

Virüs veya Truva atı gibi kötücül yazılımları görüntülere enjekte etmek veya yazılımın güncel olmayan ve savunmasız sürümlerini çalıştırmak zehirli görüntü sorununa neden olabilir.

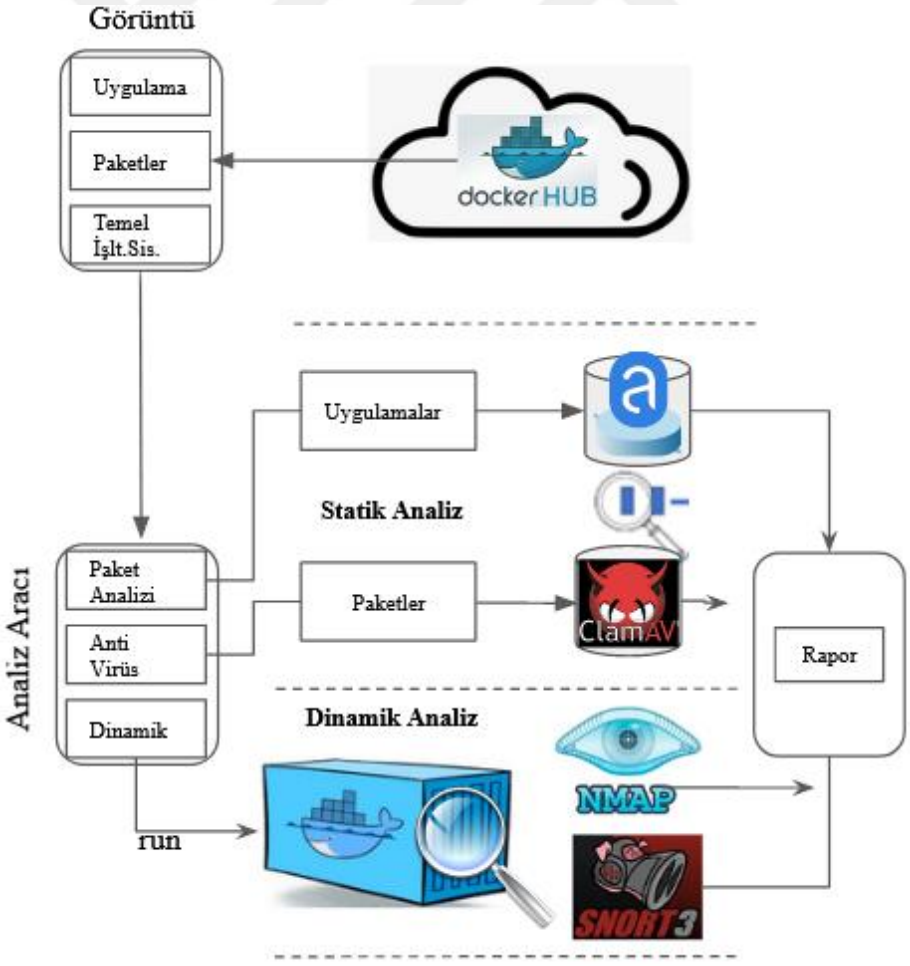
İmajlarda güncellenmemiş yazılım kullanımı kullanıcı için büyük bir risk teşkil etmektedir. Saldırganlar zafiyet barındıran yazılım paketlerini kullanarak sisteme erişim sağlayabilirler.

Kripto madenciliği saldırılarında, ana makine kaynakları kötücül yazılım türevleri aracılığıyla Ethereum ve Monero gibi sanal para madenciliği için kullanılmakta ve saldırganlara ait kripto cüzdanlara aktarılmaktadır.

Servisler ağ trafiğini dinleyen ve gerektiğinde cevap veren programlardır. FTP sunucuları, web sunucuları, dosya sunucuları, e-posta ve proxy sunucuları gibi servisler açık portlar aracılığıyla ana makineye doğrudan erişime izin vermektedir. Gereksiz ve güvenli olmayan hizmetler, saldırganlar için açık kapılardır. Ana makine üzerinde çalışıtırılan hizmet sayısındaki artış, atak yüzeyini genişletecek ve saldırganların sisteme erişim için farklı saldırı vektörlerini istismar etmesine olanak tanıyacaktır.

5. DOCKER GÖRÜNTÜ GÜVENLİĞİ MODELİ

Bu bölümde Docker görüntülerinin güvenlik kontrollerinin yapılarak çalıştırılması için görüntüler üzerinde statik ve dinamik analiz olmak üzere iki safhadan oluşan bir analiz modeli oluşturulmuştur. Güvenlik analizlerinin yapılabilmesi için oluşturulan model Şekil 5.1'de gösterilmiştir. Statik analiz safhasında; görüntüler çalıştırılmadan önce içerisinde yer alan paketlere ilişkin zafiyet taraması ve anti virüs taraması gerçekleştirilmektedir. Modelin dinamik analiz safhasındaysa; kapsayıcı çalıştığında saldırgan tarafından istismar edilebilecek portların taraması ve kapsayıcının ağ trafik analizi yapılmaktadır. Docker görüntüleri üzerinde yapılan analiz sonrası kullanıcıya görüntülerin güvenliğine ilişkin bir rapor sunulmaktadır.



Şekil 5.1 : Docker görüntü ve kapsayıcı güvenlik analiz modeli

5.1 Statik Analiz

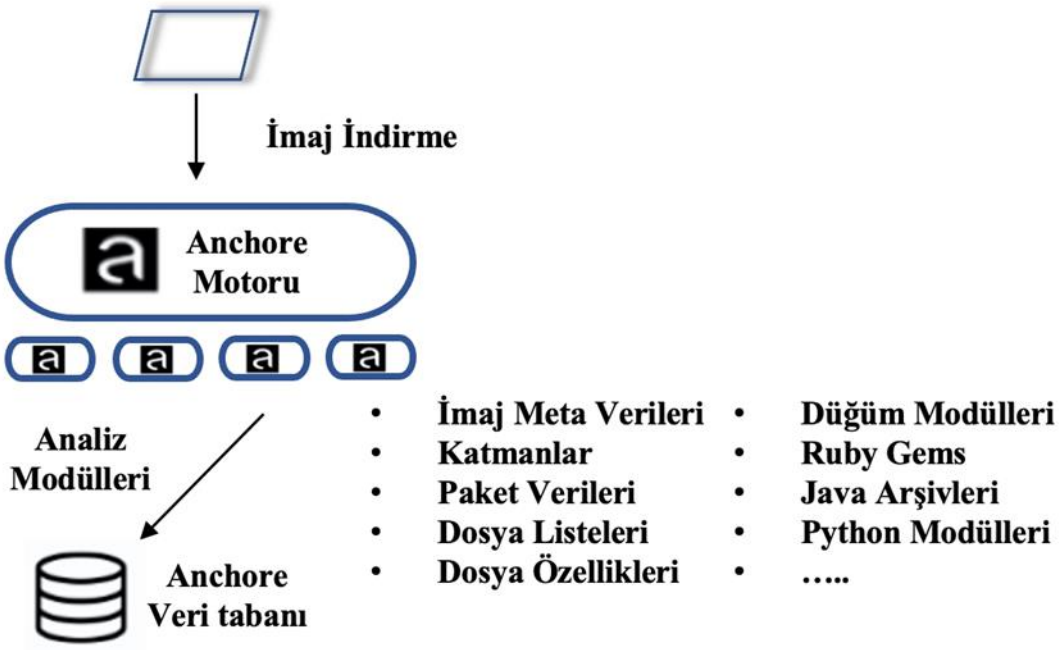
Statik analiz ile yazılım; test veya gerçek ortamda çalıştırılmadan önce içerisinde bulunan paket, kütüphane vb. gibi veriler analiz edilmektedir. Statik analiz kontrolüyle, analizi yapılan görüntüler içerisinde zararlı olarak değerlendirilebilecek metinsel ifadeler, fonksiyonlar ve hash değerleri gibi veriler üzerinde anormallikler tespit edilebilmektedir. Bir Docker görüntüsünün statik analizi, görüntü çalışmadan önce güvenlik zafiyetlerinin ve açıklıklarının görüntü çalıştırılmadan önce kontrol edilmesidir. Statik analizde, öncelikle bilinen güvenlik açıklarının kapsamlı bir kataloğu olan CVE (Common Vulnerabilities and Exposures)'den faydalanılmaktadır. CVE sistemi, bilinen güvenlik açıkları için bir referans yöntemi sağlamaktadır. CVE, ABD İç Güvenlik Bakanlığına bağlı Siber Güvenlik ve Altyapı Güvenliği Ajansı'nın finansmanıyla MITRE şirketi tarafından yönetilmektedir. CVE bilgisi, zaafiyete ilişkin teknik detayları, zaafiyetin yaratacağı riskleri, etkileri ve zaafiyetin giderilmesi hakkında bilgi içermemektedir. Bir CVE kodu, benzersiz bir tanımlayıcıyla belirtilir ve her tanımlayıcının standartlaştırılmış bir açıklaması bulunur. CVE'nin önem derecesi, Ortak Güvenlik Açığı Puanlama Sistemi (CVSS) kullanılarak derecelendirilir. Bu anlamda CVE bilgi sistemlerinin değerlendirilmesine ve güvenlik açıklarının belirlenmesinde uluslararası bir tanımlama sistemi ortaya koyarak açıklık yönetimine yardımcı olmaktadır.

5.1.1 Docker Kapsayıcı Paket Analizi

Modelde; görüntülerin statik analizi kapsayıcı olarak çalıştırılan Anchore aracıyla yapılmaktadır. Anchore aracı, kapsayıcı görüntülerinin incelenmesi, analizi ve sertifikasyonu için merkezi bir hizmet sağlayan açık kaynaklı projedir. Anchore, bağımsız olarak veya Kubernetes, Docker Swarm, Rancher, Amazon ECS ve diğer kapsayıcı platformları üzerinde çalıştırılabilen bir kapsayıcı görüntüsü olarak kullanılmaktadır [38]. Kullanıcıların Anchore aracı üzerinde güvenlik politikalarını özelleştirmesi, uygulamalar üzerinde sıkı bir kontrole sahip olmalarına ve NIST 800-190 (Application Container Security Guide) standartının karşılanmasına imkan sağlamaktadır [39]. Anchore aracı ilk kez çalıştırıldığında, güvenlik açığı verilerinin senkronize edilmesi, ağ hızına bağlı olarak on dakika ve üzerinde bir zaman içerisinde gerçekleşmektedir. Anchore aracı ile yapılan taramalarda, doğru sonuçlara ulaşılması için güvenlik açıklıklarına ait bilgilerin güncellenmesi gerekmektedir. Anchore

aracının senkronize ettiği verileri komut betikleriyle görmek mümkündür. Aracın çalışması için bağımlı olduğu sistem bir PostgreSQL veritabanıdır (9.6+) [40].

Anchore aracı, Docker görüntüsünde işletim sistemi haricindeki paketleri ve paket bağımlılıklarını taraması, güvenlik zafiyetlerini farklı kaynaklardan alarak CVE tabanlı tarama gerçekleştirmesi ve zafiyet güncellemelerini sıklıkla yapması nedeniyle modelde tercih edilmiştir. Docker Hub üzerinden indirilen Docker görüntüleri, CVE taramalarının yapılarak incelenmesi için kapsayıcı olarak çalıştırılan Anchore aracına gönderilmektedir. Görüntü analiz için Anchore motoruna gönderildiğinde, Anchore, Docker kayıt defterinden görüntüye ilişkin meta verileri almaya çalışır. Meta verileri alınan görüntü indirilerek analiz için kuyruğa alınır. Analiz esnasında görüntüye ait her paket, yazılım kütüphanesi ve dosya incelenmektedir. Anchore aracına ait motorlar vasıtasıyla yapılan analizler çıktı olarak kullanıcıya sunulmaktadır. Şekil 5.2'de Anchore aracının çalışma yapısı gösterilmiştir. Anchore açık kaynak aracına ait docker-compose.yml dosyası Ek-1'de sunulmuştur.



Şekil 5.2 : Anchore analiz modülleri

5.1.2 Docker Kapsayıcı Anti Virüs Taraması

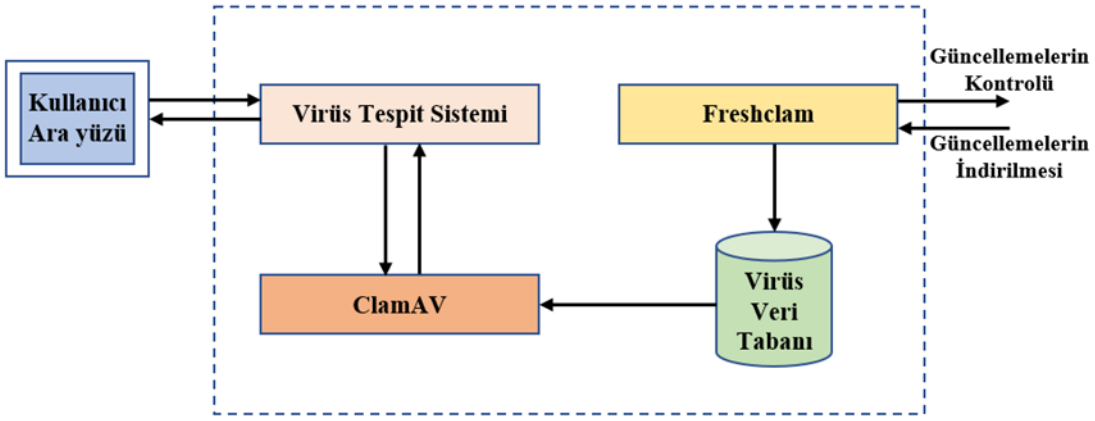
Docker görüntü güvenliğinin sağlanmasında dikkat edilmesi gereken bir diğer önemli tehdit, kötü amaçlı yazılımlardır. Kötü amaçlı yazılımlar, sistem sahibinin bilgisi ve rızası olmadan bir bilgisayar sistemine sızmayı amaçlamaktadır. Kötü amaçlı yazılım yıkıcı ve müdahaleci etkisiyle sistemlerin çalışma yapısını bozmaktadır. Virüsler,

solucanlar, truva atları, rootkitler ve casus yazılımlar farklı biçim ve türde kötü amaçlı programlar olup, yayılma yöntemine ve geliştirilme amacına göre sınıflandırılmaktadır. Örneğin kötü amaçlı yazılımlar aracılığıyla, istenmeyen reklamların başlatılması, kullanıcının bilgisi olmadan e-posta gönderilmesi, sistemler üzerinde casusluk yapılması, sistemin içeriden yok edilmesi gibi eylemler gerçekleştirilebilir.

Kötü amaçlı yazılım tespitinde, imza ve davranış tabanlı sistemler ile kum havuzu kullanılmaktadır. İmza tabanlı sistemlerde, kötü amaçlı yazılımı ve türünü tanımlamak için bir imza veya bit dizisi kullanılmaktadır. Kötücül yazılımlara ait imzalar, bulut üzerinde bulunan veri tabanlarında toplanmaktadır. Siber güvenlik firmaları ve araştırmacılar tarafından herhangi bir yeni kötücül yazılım tespit edildiğinde, zararlıya ait imza, anti virüs veri tabanına eklenmektedir. Bir sistem üzerindeki anti virüs taramasında, taranan dosyalara ait imzalar veri tabanındaki imzalar ile karşılaştırılarak kontrol edilir ve herhangi bir eşleşme durumunda, dosya kötü amaçlı yazılım olarak tanımlanır. Kötü amaçlı yazılım barındırdığı tespit edilen dosyaların, bilgisayar sistemleri üzerinde çalıştırılmasına izin verilmez. İmza tabanlı zararlı tespit sistemleri, erişilebilir, kolay çalıştırılabilir ve hızlı bir yöntem olması nedeniyle tercih edilmektedir.

Açık kaynak Docker Hub görüntü paylaşım platformunda bulunan görüntülere, virüs, truva atı, arka kapılar, web kabuğu (web shell), kripto para madenciliği, fidye yazılımı gibi kötücül dosyalar eklenebilir. Zararlı yazılım barındıran görüntüler, konak işletim sisteminde çalıştığında ciddi güvenlik sorunlarına neden olabilir. Bu nedenle, Docker görüntülerinde bulunabilecek zararlı yazılımlara karşı anti virüs taramasının yapılması oldukça önemlidir.

ClamAV, Unix işletim sistemi üzerinde C programlama dilinde geliştirilmiş açık kaynak kodlu bir anti virüs programıdır. ClamAV kötü amaçlı yazılımları tespit etmek için metin (string) tabanlı imzaları kullanmaktadır. ClamAV virüs tespit programı, virüs veritabanı ve freshclam aracıyla çalışır. Freshclam, ClamAV'ın veri tabanını güncellemesine yardımcı olmaktadır [41].



Şekil 5.3 : ClamAV mimarisi [42]

Modelde; ClamAV Docker kapsayıcı olarak çalıştırılmaktadır. ClamAv Docker görüntüsünün oluşturulmasına yönelik hazırlanan Dockerfile dosyası Ek-2'de sunulmuştur. Docker Hub üzerinden indirilen görüntüler, modelde sıkıştırılarak tar dosya tipinde kaydedilmektedir. Kapsayıcı olarak çalıştırılan ClamAV içerisinde, kaydedilen tar uzantılı dosyalar açılarak anti virüs taraması yapılmakta ve tarama sonuçları bir rapor olarak sunulmaktadır.

5.2 Dinamik Analiz

Dinamik Analiz; bir programın, gerçek zamanlı olarak çalıştırılarak etki alanı, gerçekleştirdiği işlemler, olası komuta kontrol sunucularıyla yapılan iletişim ve IP adresleri gibi bilgilerin elde edilmeye çalışıldığı analiz işlemidir. Bu analiz yönteminde test edilecek yazılım çalıştırılarak, dosya dizinlerindeki hareketleri, ağ aktiviteleri, DNS sorguları, kayıt defteri üzerindeki aktiviteleri gibi durumlar incelenir. Kapsayıcı güvenliğinde ise dinamik analiz ile bir kapsayıcının davranışı gözlemlenmektedir. Dinamik analizin statik analize kıyasla daha uzun sürmesine rağmen ortaya çıkardığı sonuçlar daha anlamlı olabilmektedir. Çalışan kapsayıcıların davranışları, CPU, bellek kullanımı, ağ trafiği veya sistem çağrıları gibi parametrelerde ortaya çıkmaktadır. Dinamik analizde kullanılan metotlara; görüntü çalıştırmadan önce veya sonra port taramaları, proses izleme, kayıt defteri değişiklikleri ve ağ trafiğinin izlenmesi örnek olarak verilebilir. CPU, bellek kullanımı ve ağ trafiğinin artması, kapsayıcı istismar tespitinde bir gösterge olarak kullanılabilir, ancak tek başına kaynakların kullanımındaki bu artış, kapsayıcı içerisinde kötücül bir yazılım bulunduğu anlamına gelmemektedir. Kapsayıcı görüntülerinin ağ yalıtımı yapılmadan konak işletim sisteminin de dahil olduğu ağ üzerinde çalıştırılmaları

risklidir. Sistemdeki diğer hizmetlerin ve kaynakların çalıştırılacak olan kapsayıcıdan etkilenmemesi için kapsayıcıların yalıtılmış bir ortamda çalıştırılması gerekmektedir. Docker görüntü güvenliğine yönelik yapılan çalışmalar, dinamik analizin Docker kapsayıcılara ilişkin kötü niyetli davranışları tespit etmede etkili olduğunu göstermektedir. Kelly ve diğerleri tarafından yapılan çalışmada, kötücül görüntülerin genellikle komuta ve kontrol sunucuları ile iletişime geçmek için ssh tünelleri oluşturan bash betikleri içerdiği, ayrıca kötücül görüntülerin ilave binaryleri indirmeye ve shellcode yüklemeye çalıştığı tespit edilmiştir [36].

5.2.1 Docker Kapsayıcı Ağ Trafik Analizi

Docker kapsayıcılar internete konak işletim sistemi aracılığı ile bağlanırlar. Kötücül Docker görüntülerinin dinamik analiz ile tespitinde, kapsayıcının yapmış olduğu DNS sorguları ve hedef IP bağlantılarının, birçok açık kaynaktan beslenen kara liste veri tabanında sorgulanması gerekmektedir.

Snort, bir ağda gerçek zamanlı trafik ve paket analizi gerçekleştirebilen açık kaynak kodlu bir ağ saldırı tespit ve önleme sistemidir. Saldırı tespit sistemi, temel olarak ağ trafiğindeki paketlerin tespiti ve izlenmesi için kullanılan bir yazılımdır. Snort, ağ üzerinde gerçekleştirilen olası kötücül faaliyetleri tespit etmek için protokol ve imza yöntemlerini birleştiren kural tabanlı bir dil kullanmaktadır. Kural tabanlı bir dil kullanılması sayesinde karşılaşılan yeni saldırılara ait kurallar yazılarak veri tabanına eklenebilmektedir. Ağ trafiğinde saldırı emaresi gösteren anormal veri paketleri tespit edildiğinde, sistem üzerinde uyarı oluşturmaktadır. Snort aracı, protokol analizi, ağ trafiğinde içerik araması/eşleştirmesi yapabilmekte, arabellek taşmaları, arka kapı bağlantı noktalarını, CGI (Common Gateway Interface) saldırıları, SMB taramaları gibi çeşitli saldırıları tespit edebilmektedir.

Modelde, dinamik analizi yapılacak olan her bir kapsayıcı için konak işletim sisteminde diğer ağlardan yalıtılmış farklı bir ağ oluşturulmaktadır. Kapsayıcılar kendilerine özgü oluşturulmuş bu ağ üzerinde doksan saniye çalıştırılmaktadır. Kapsayıcıların çalıştırıldıkları süre boyunca oluşturduğu ağ trafiği “tcpdump” aracı ile kaydedilmektedir. Kaydedilen ağ trafik dosyası analiz için kapsayıcı olarak çalıştırılan Snort3 aracına gönderilmekte ve sonuçlar kullanıcıya bir rapor olarak sunulmaktadır. Kapsayıcı olarak çalıştırılan Snort3 Docker görüntüsüne ait gerekli “dockerfile” dosyası Ek-3’te sunulmuştur.

5.2.2 Docker Kapsayıcı Port Taraması

Modelde dinamik analizi yapılan Docker görüntüleri, yalıtılmış bir ağ üzerinde doksan saniye çalıştırılmış olup, çalıştırıldıktan sonra servislerin sağlıklı bir şekilde başlatılması için port tarama işlemi on saniye sonra başlatılmıştır.

Modelde; port taraması için açık kaynak kodlu nmap tarama aracı kullanılmaktadır. Konak işletim sisteminde diğer ağlardan yalıtılmış farklı bir ağ üzerinde çalışmaya başlayan her bir kapsayıcı için port taraması gerçekleştirilmektedir. Yapılan port tarama sonuçları bir çıktı olarak kullanıcıya sunulmaktadır.

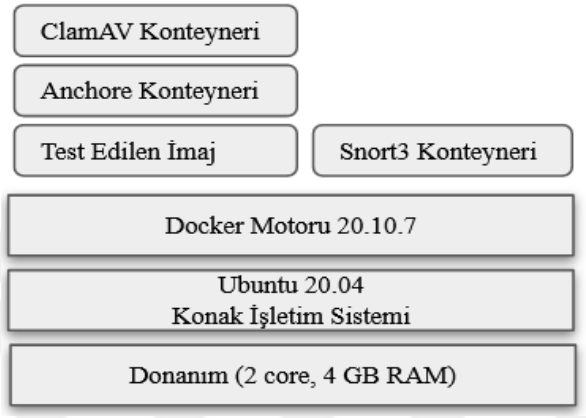
5.3 Önerilen Modelin Değerlendirilmesi

Bu bölümde önerilen modelin değerlendirilmesi için Docker Hub görüntü paylaşım platformu üzerinde yer alan 20 adet görüntü modelin değerlendirilmesi için seçilmiştir. Görüntü seçiminde indirilme sayısı ve görüntü ile sunulan hizmet türü seçim kriteri olarak esas alınmıştır. Ayrıca modelin işlerliğinin test edilebilmesi için zararlı ağ trafiğini simüle eden "Network Flight Simulator" aracı osengul/maliciousdocker görüntüsü haline getirilmiştir. Seçilen görüntüler ve bu görüntülere ait bilgiler, Çizelge 5.1'de gösterilmektedir.

Çizelge 5.1 : Seçilen görüntü bilgileri

S. No.	Görüntü Adı	Türü	Sürümü	Yüklenme Tarihi
1	nginx	Web Sunucu	1.21.0	25.05.2021
2	postgresql	Veritabanı	13	17.08.2021
3	ubuntu	İşletim Sistemi	20.04	23.04.2021
4	buamod/eicar	Anti virüs Test	Yok	01.06.2018
5	docheck/health	İmajı Kapsayıcı Monitörleme	1.13.1	02.04.2018
6	imiell/bad-dockerfile	Zafiyetli Paket Barındıran Test İmajı	17.06.1	01.11.2017
7	osengul/malicious	Zararlı Ağ Trafiği Simülasyon İmajı	1.0	06.09.2021
8	ghost	Blog Oluşturma Platformu	4.19.1	18.10.2021
9	maite/application	Spor Koçu Proje İmajı	Yok	28.03.2017
10	cassandra	NoSQL Veritabanı	4.0.1	16.10.2021
11	gazgeek/springboot-helloworld	Uygulama Geliştirme Çerçevesi	Yok	11.03.2015
12	docheck/ax	Belirtilmemiş	Yok	02.04.2018
13	ubvntu/utnubu	Belirtilmemiş	Yok	11.02.2021
14	osengul/flag	Ctf Oyunlaması	Yok	09.12.2021
15	tempsbro/tempsbro	Belirtilmemiş	Yok	06.10.2018
16	tester122/cehdumpsv11	CEHv11 Hazırlık Soruları	Yok	13.12.2021
17	covid29/vaccine4all	Web Sunucu	Yok	16.12.2021
18	ubvntu/vbuntu	Belirtilmemiş	Yok	05.09.2021
19	antiviruss/scanner	Anti virus	Yok	20.12.2021
20	covid29/commandcontrol	Web Sunucu	Yok	16.12.2021

Test ortamı için Intel i9-9900K işlemcili, 16 GB ana bellek, NVIDIA GeForce RTX 2070 Super ekran kartı donanım özelliklerine sahip Windows 10 Pro (versiyon 21H1-19043.1165) işletim sistemi üzerindeki ana makinede kurulu WMware Workstation (versiyon 16.1.2 build-17966106) sanallaştırma uygulaması kullanılmıştır. Önerilen model, 4 GB ana bellek ve iki çekirdek tahsis edilen Ubuntu 20.04 konak işletim sisteminde test edilmiştir. Önerilen modelin fiziksel topolojisi Şekil 5.4'te gösterilmiştir.



Şekil 5.4 : Önerilen model test ortamı fiziksel topolojisi

Docker Hub'da en güvenli görüntü türü olarak değerlendirilen resmi görüntülerin %45,9'u en az bir kritik veya yüksek dereceli güvenlik açığı barındırmaktadır [4]. Modelin değerlendirilmesi için seçilen görüntülere ait statik analiz sonuçları Çizelge 5.2'de gösterilmiştir.

Çizelge 5.2 : Görüntülere ait statik analiz ve anti virüs sonuçları

S. No.	Görüntü Adı	CVE			Anti Virüs
		Orta	Yüksek	Kritik	
1	nginx	1	5	2	- [a]
2	postgresql	1	3	2	-
3	ubuntu	-	-	-	-
4	buamod/eicar	3	5	1	1
5	docheck/health	3	5	1	1
6	imiell/bad-dockerfile	671	270	48	-
7	osengul/malicious	-	-	-	-
8	ghost	13	11	3	-
9	maite/application	630	583	508	-
10	cassandra	-	1	-	-
11	gazgeek/springboot-helloworld	923	135	81	-
12	docheck/ax	4	12	3	1
13	ubvntu/utnubu	18	32	5	1
14	osengul/flag	4	20	3	1
15	tempsbro/tempsbro	459	30	-	1
16	tester122/cehdumpsv11	15	17	4	3
17	covid29/vaccine4all	844	384	-	1
18	ubvntu/vbuntu	22	50	5	1
19	antiviruss/scanner	844	384	-	1
20	covid29/commandcontrol	7	26	6	1

[a] Tespit edilmemiştir.

Model tarafından statik analizi yapılan açık kaynak kodlu web sunucu yazılımı "nginx" Docker görüntüsü; Docker Hub platformu üzerinden bir milyardan fazla kez indirilmiştir. İndirilme sayısı esas alınarak seçilen nginx Docker görüntüsü üzerinde gerçekleştirilen statik analizde iki tane CVE-2021-3711 kodlu kritik açıklık tespiti ile iki tane CVE-2021-22924, iki tane CVE-2021-3712 ve bir tane CVE-2021-3712 olmak üzere toplamda beş tane yüksek dereceli açıklık tespiti yapılmıştır. Ayrıca nginx Docker görüntüsünde, CVE-2011-3389 kodlu bir tane orta dereceli açıklık tespiti yapılmıştır. Bir milyardan fazla indirilen nginx görüntüsünde gerçekleştirilen ClamAV anti virüs taramasında, yirmi iki dosya taranmış ve herhangi bir kötücül yazılım imzasına rastlanılmamıştır.

Bir milyardan fazla kez indirilen açık kaynak kodlu postgresql veritabanı Docker görüntüsüne ilişkin gerçekleştirilen statik analizde; iki tane CVE-2021-3711 kodlu kritik dereceli açıklık ile iki tane CVE-2021-3712 kodlu ve bir tane CVE-2021-30535 kodlu olmak üzere üç tane yüksek dereceli açıklık tespiti yapılmıştır. Ayrıca postgresql Docker görüntüsünde CVE-2011-3389 kodlu bir tane orta dereceli açıklık tespiti yapılmıştır. Açık kaynak kodlu olması ve performans etkin olması nedeniyle indirilme sayısı bir milyarı geçen postgresql görüntüsüne ait anti virüs taramasında, kırk üç dosya taranmış ve herhangi bir kötücül yazılım tespiti yapılmamıştır.

Linux çekirdeği temel alınarak geliştirilen açık kaynak kodlu bir işletim sistemi olan Ubuntu, model üzerinde statik analiz testlerine tabi tutulmuştur. Bir milyardan fazla indirilen Ubuntu 20.04 Docker görüntüsüne ait kritik, yüksek ve orta kritiklik dereceli açıklık tespiti yapılmamıştır. Görüntünün statik analizinde otuz altı tane düşük kritik dereceli açıklık tespiti yapılmıştır. Ubuntu görüntüsüne ait on üç dosya üzerinde yapılan anti virüs taramasında kötücül yazılım tespit edilmemiştir.

Modelin anti virüs tarama etkinliğinin ve ClamAv anti virüs performansının test edilmesi amacıyla; Avrupa anti virüs araştırmacıları tarafından, antivirüs firmaları ile anlaşılarak hazırlanan EICAR (European Institute for Computer Antivirus Research) Test dosyası barındıran ve Docker Hub üzerinden yaklaşık üç bin kez indirilen "buamod/eicar" Docker görüntüsü kullanılmıştır. Görüntüye yönelik yapılan statik analizde, 1 tane CVE-2018-1000517 kodlu kritik açıklık, CVE-2018-1000500 kodlu ve CVE-2018-20679 kodlu iki tane yüksek dereceli açıklık tespiti ile CVE-2018-0732 kodlu iki, CVE-2020-28928 kodlu bir tane orta dereceli açıklık tespiti yapılmıştır.

ClamAv anti virüs testinde, on üç dosya taranmış ve bu dosyalar arasında Eicar test dosyası başarılı bir şekilde tespit edilmiştir.

Kripto madenciliği yaptığı bilinen [43], Docker Hub üzerinde yüz binden fazla indirilen "docheck/health" görüntüsünün statik analizinde; CVE-2018-1000517 kodlu bir tane, CVE-2019-14697 kodlu iki tane olmak üzere üç tane kritik dereceli, CVE-2018-0732 kodlu iki tane, CVE-2018-20679 kodlu bir tane, CVE-2019-5747 bir tane olmak üzere 5 tane yüksek dereceli açıklık tespiti yapılmıştır. Ayrıca "docheck/health" görüntüsü üzerinde CVE-2018-0495 kodlu iki tane, CVE-2020-28928 kodlu bir tane olmak üzere toplamda üç tane orta dereceli açıklık tespiti yapılmıştır. Görüntünün ClamAV anti virüs testinde, on dokuz dosyanın taraması yapılarak, "Multios.Trojan.CryptocoinMiner-6448864-1" kripto madencilik zararlı yazılımı tespit edilmiştir.

Zafiyetli paket barındırdığı bilinen [44] ve Docker Hub paylaşım platformundan elli binin üzerinde indirilen "imiell/bad-dockerfile" görüntüsü üzerinde 48 kritik, 270 yüksek ve 671 adet orta dereceli zafiyet, statik analiz ile tespit edilmiştir. ClamAV anti virüs testinde altmış dört dosya taranmış olup görüntü üzerinde herhangi bir zararlı yazılıma rastlanılmamıştır.

Alphasoc firması tarafından siber güvenlik operasyon merkezlerinde görevli personelin eğitimi amacıyla geliştirilen "Network Flight Simulator" aracı, zararlı olduğu bilinen alan adlarına DNS sorguları gerçekleştirmekte ve zararlı ağ trafiğini taklit etmektedir. "Network Flight Simulator" aracının kapsayıcı görüntüsü haline getirilmesi amacıyla Dockerfile dosyası oluşturulmuş ve osengul/maliciousdocker ismi ile Docker Hub paylaşım platformu üzerinde tüm kullanıcıların paylaşımına sunulmuştur. Modelde statik analizi yapılan osengul/maliciousdocker görüntüsü içerisinde yirmi beş dosya içerisinde zafiyetli paket ve kötücül yazılım bulunmamıştır.

Statik analizi gerçekleştirilen bir diğer Docker görüntüsü; Docker Hub üzerinden yüz binden fazla indirilen açık kaynak kodlu blog platformu olan ghost'dur. "ghost" görüntüsünün statik analizinde, iki tane CVE-2021-41720 kodlu, bir tane GHSA-4c7m-wxvm-r7gc kodlu üç tane kritik dereceli, on bir tane yüksek dereceli ve on üç tane orta dereceli açıklık tespiti yapılmıştır. "ghost" görüntüsüne ait otuz bir adet dosya üzerinde yapılan ClamAV anti virüs taraması sonucu kötücül yazılım tespiti edilmemiştir.

Docker Hub'da "maite" kullanıcı ismiyle, beş yıl önce yüklenen "maite/application" Docker görüntüsü beş yüz binden fazla indirilmiştir. "maite/application" Docker görüntüsü kullanıcılara spor koçluğu yapmak amacıyla oluşturularak kullanıcalara sunulmuştur. Görüntünün statik analizinde beş yüz sekiz tane kritik, beş yüz seksen üç tane yüksek ve altı yüz otuz tane orta dereceli açıklık tespiti yapılmıştır. Görüntünün anti virüs taraması kapsamında yirmi sekiz adet dosya taranmış ve herhangi bir zararlı yazılım tespit edilmemiştir.

Açık kaynak kodlu NoSQL ve dağıtık mimariye sahip "cassandra" veritabanı Docker görüntüsünün statik analizi yapılmıştır. Docker Hub görüntü paylaşım platformunda yüz milyondan fazla indirilen "cassandra" görüntüsü üzerinde gerçekleştirilen statik analiz sonucunda bir tane kritik dereceli CVE-2018-8016 kodlu ve yüksek dereceli CVE-2020-17516 kodlu toplam iki adet açıklık tespiti yapılmıştır. Anti virüs taraması kapsamında görüntüye ait toplam otuz bir dosya üzerinde inceleme gerçekleştirilmiş olup herhangi bir zararlı yazılım tespit edilmemiştir.

Spring tabanlı uygulama geliştirmek için kullanılan "gazgeek/springboot-helloworld" Docker görüntüsü gazgeek kullanıcısı tarafından yaklaşık altı sene önce kullanıcılara sunulmuş olup beş yüz binden fazla indirilmiştir. Görüntünün statik analizi sonucunda seksen bir tane kritik, yüz otuz beş tane yüksek ve dokuz yüz yirmi üç tane orta dereceli açıklık tespiti yapılmıştır. Görüntünün anti virüs taramasında zararlı yazılım tespiti yapılmamıştır.

"docheck/ax" görüntüsü Docker Hub görüntü paylaşım platformundan yüz binden fazla kez indirilmiştir. Bu görüntü üzerinde yapılan staik analiz sonucunda, CVE-2018-1000517, CVE-2019-14697, CVE-2019-14697 kodlu üç tane kritik dereceli açıklık tespiti yapılmıştır. Ayrıca görüntü üzerinde on iki adet yüksek ve dört adet orta dereceli açıklık tespiti yapılmıştır. Görüntünün ClamAV anti virüs testinde, on dokuz dosyanın taraması yapılarak, "Multios.Trojan.CryptocoinMiner-6448864-1" kripto madencilik zararlı yazılımı barındırdığı tespit edilmiştir.

Docker Hub üzerinde "ubvntu" kullanıcısına ait repolarda bulunan görüntüler model tarafından analiz edilmiştir. Bu kullanıcı tarafından paylaşılan "ubvntu/utnubu" görüntüsünün statik analizinde; iki tane CVE-2021-22945, iki tane CVE-2021-3711 ve bir tane CVE-2021-36159 toplamda beş adet kritik dereceli açıklık ile otuz iki tane yüksek ve on sekiz tane orta dereceli açıklık tespiti yapılmıştır. Görüntünün ClamAV

anti virüs testinde, on altı dosyanın taraması yapılarak, "Unix.Trojan.Generic-9909259-0" kripto madencilik zararlı yazılımı barındırdığı tespit edilmiştir.

Bayrak yakalama yarışması için oluşturulan "osengul/flag" Docker görüntüsünün statik analizinde; CVE-2021-36159 kodlu bir tane, CVE-2021-3711 kodlu iki tane olmak üzere üç adet kritik dereceli, yirmi tane yüksek dereceli ve dört tane orta dereceli açıklık tespiti yapılmıştır. Görüntünün ClamAV anti virüs testinde, yirmi beş dosyanın taraması yapılarak, "Win.Trojan.MSShellcode-6" zararlı yazılımı barındırdığı tespit edilmiştir.

Docker Hub üzerinden yüz binden fazla kez indirilen ve analizi yapılan bir diğer Docker görüntüsü "tempsbro/tempsbro" üzerinde; otuz tane yüksek dereceli, dört yüz elli dokuz tane orta dereceli açıklık tespiti yapılmıştır. Görüntünün ClamAV anti virüs testinde, kırk adet dosyanın taraması yapılarak, "Multios.Coinminer.Miner-6781728-2" kripto madencilik zararlı yazılımı barındırdığı tespit edilmiştir.

Kullanıcılara, üzerinde "Certified Ethical Hacker" sertifikasyon sınavına ait çalışma sorularının olduğunu düşüntürtecek şekilde isimlendirilerek Docker Hub üzerinde paylaşılan "tester122/cehdumpsv11" görüntüsünün analizinde; CVE-2016-2148, CVE-2016-9841, CVE-2016-9843, CVE-2018-1000517 kodlu dört tane kritik, on yedi adet yüksek ve on beş adet orta kritik dereceli açıklık tespiti yapılmıştır. Görüntünün ClamAV anti virüs testinde, on dokuz dosyanın taraması yapılarak, üç tane "Win.Trojan.Poison-8692" zararlı yazılımı barındırdığı tespit edilmiştir.

Docker Hub üzerinde "covid29" kullanıcı adıyla oluşturulan repolar, model tarafından analiz edilmiştir. Bu kullanıcı tarafından paylaşılan "covid29/vaccine4all" görüntünün statik analizinde; üç yüz seksen dört adet yüksek ve sekiz yüz kırk dört adet açıklık tespiti yapılmıştır. Görüntünün ClamAV anti virüs testinde, seksen dosyanın taraması yapılarak, bir adet "Unix.Trojan.WINNTI-7793130-1" zararlı yazılımı barındırdığı tespit edilmiştir.

Docker Hub üzerinde "ubvntu" kullanıcısı tarafından paylaşılan bir diğer Docker görüntüsü "ubvntu/vbuntu" üzerinde yapılan analizde; iki tane CVE-2021-22945, iki tane CVE-2021-3711 ve bir tane CVE-2021-36159 toplamda beş adet kritik dereceli açıklık ile elli tane yüksek ve yirmi iki tane orta dereceli açıklık tespiti yapılmıştır. Görüntünün ClamAV anti virüs testinde, on dosyanın taraması yapılarak, bir adet

“Multios.Coinminer.Miner-6781728-2” kripto madencilik zararlı yazılımı barındırdığı tespit edilmiştir.

Docker Hub üzerinde “antiviruss/scanner” adıyla paylaşılan görüntü üzerinde yapılan analizde; üç yüz seksen dört adet yüksek ve sekiz yüz kırk dört adet açıklık tespiti yapılmıştır. Kullanıcılara bir anti virüs tarama hizmeti sunuyor gibi isimlendirilerek yüklenen bu görüntünün ClamAV anti virüs testinde, yüz iki dosya taraması gerçekleştirilmiş ve bir adet “Js.Coinminer.Generic-6836639-1” kripto madencilik zararlı yazılımı barındırdığı modelimiz tarafından tespit edilmiştir.

Docker Hub üzerinde “covid29” kullanıcısı tarafından paylaşılan bir diğer Docker görüntüsü “covid29/commandcontrol” üzerinde yapılan analizde; CVE-2016-2148, CVE-2016-9841, CVE-2016-9843, CVE-2017-6349, CVE-2017-6350, CVE-2018-1000517 kodlu altı tane kritik dereceli açıklık tespit edilmiştir. Ayrıca bu görüntü üzerinde yirmi altı tane yüksek ve yedi tane orta dereceli açıklık tespiti yapılmıştır. Görüntünün ClamAV anti virüs testinde, on yedi dosyanın taraması yapılarak, bir adet “Legacy.Trojan.Agent-379” zararlı yazılımı barındırdığı tespit edilmiştir.

Model tarafından statik analiz sonrası sınırlı yalıtılmış bir ağda test edilen görüntülerin dinamik analiz sonuçları Çizelge 5.3’te gösterilmiştir.

Çizelge 5.3 : Görüntülere ait dinamik analiz sonuçları

S. No.	Görüntü Adı	Açık Port Taraması	Ağ Trafik Analizi
1	nginx	TCP/80	- [b]
2	postgresql	* [c]	*
3	ubuntu	-	-
4	buamod/eicar	-	-
5	docheck/health	-	-
6	imiell/bad-dockerfile	-	-
7	osengul/malicious	-	Şüpheli DNS Sorgusu
8	ghost	-	-
9	maite/application	-	-
10	cassandra	TCP 7000	-
11	gazgeek/springboot-helloworld	TCP 8080	-
12	docheck/ax	-	-
13	ubvntu/utnubu	-	-
14	osengul/flag	TCP 22	-
15	tempsbro/tempsbro	-	-
16	tester122/cehdumpsv11	-	-
17	covid29/vaccine4all	TCP 80	
18	ubvntu/vbuntu	-	-
19	antiviruss/scanner	TCP 80	-
20	covid29/commandcontrol	-	Kara Liste Ip adreslerine Erişim

[b.] Tespit edilmemiştir.

[c.] Özel parametrelerle çalıştırılması gerekmektedir.

Modelde dinamik analizi yapılan nginx Docker görüntüsünün port taraması sonucu beklendiği gibi TCP/80 portunun açık olduğu tespit edilmiştir. Çalıştığı süre boyunca kaydedilen ağ trafiği, analiz için kullanılan Snort3 aracına gönderilmiş olup herhangi bir zararlı ağ trafiği tespiti yapılmamıştır.

Postgresql veri tabanı varsayılan olarak TCP/5432 portunu kullanmaktadır. Ancak postgresql görüntüsünün çalışması için özel parametreler girilerek süper kullanıcı (superuser) parolası ile aktif edilmesi gerekmektedir. Model üzerinde özel parametre girişi yapılamamasından dolayı açık postgresql kapsayıcısı çalıştırılamamış ve port taraması yapılamamıştır.

"buamod/eicar" Docker görüntüsü üzerinde yapılan dinamik analizde açık port tespit edilmemiş olup kaydedilan ağ trafiği üzerinde de herhangi bir olağan dışılık görülmemiştir.

Kripto madencilik zararlı yazılımı barındırdığı tespit edilen "docheck/health" ve "docheck/ax" kapsayıcısının dinamik analizinde yapılan port taraması ile açık port ve kaydedilen ağ trafiklerinin analizinde de zararlı ağ trafiği tespiti yapılmamıştır.

Zafiyetli paket barındırdığı bilinen "imiell/bad-dockerfile" Docker görüntüsü üzerinde yapılan dinamik analizde açık port ve zararlı ağ trafiği tespiti yapılmamıştır.

Modelin dinamik analiz çalışırlığının kontrolünü sağlamak maksadıyla oluşturulan "osengul/malicious" Docker kapsayıcısı içerisinde bulunan "Network Flight Simulator" aracılığıyla, bilinen zararlı alan adları için yapılan DNS istekleri, Snort3 tarafından "INDICATOR-COMPROMISE Suspicious .top dns query" gibi uyarılarla başarılı bir şekilde tespit edilmiştir. Kapsayıcının dinamik analizinde nmap aracıyla yapılan port taramasında açık port tespiti yapılmamıştır.

Açık kaynak kodlu blog oluşturma platformu olan "ghost" görüntüsünün dinamik analizinde açık port tespiti yapılmamıştır. "ghost" kapsayıcısının kaydedilen ağ trafik analizinde de herhangi bir zararlı ağ trafiği tespit edilmemiştir.

Spor koçluğu yapmak amacıyla oluşturulmuş "maite/application" Docker görüntüsünün dinamik analizinde, açık port tespiti yapılmamıştır. Kaydedilen ağ trafik analizinde de olağan dışılık görülmemiştir.

Açık kaynak kodlu NoSQL ve dağıtık mimariye sahip "cassandra" veritabanının dinamik analizinde yapılan port taramasında TCP/7000 portunun açık olduğu tespit

edilmiştir. Ağ trafiği üzerinde Snort3 ile gerçekleştirilen analizde ise herhangi bir zararlı ağ trafiği tespiti yapılmamıştır.

Spring tabanlı uygulama geliştirmek için kullanılan "gazgeek/springboot-helloworld" Docker görüntüsünün dinamik analizinde yapılan port taramasında TCP/8080 portunun açık olduğu tespit edilmiştir. Ağ trafiğinin analizinde ise olağan dışı ağ trafiği tespiti yapılmamıştır.

Kripto madencilik zararlı yazılımı barındırdığı tespit edilen "ubvntu/utnubu", "ubvntu/vbuntu" ve "tempsbro/tempsbro" kapsayıcıları ile üç tane "Win.Trojan.Poison-8692" zararlı yazılımı barındıran "tester122/cehdumpsv11" kapsayıcısının dinamik analizinde yapılan port taraması ile açık port tespit edilmemiştir. Ayrıca bu kapsayıcıların kaydedilen ağ trafiklerinin analizinde zararlı bir ağ trafiği tespiti yapılmamıştır.

Bayrak yakalama yarışması için oluşturulan "osengul/flag" kapsayıcısının dinamik analizinde TCP 22 ssh portu üzerinden "root" kullanıcı adı ve kullanıcı adıyla aynı olan "root" şifresiyle erişilebildiği tespit edilmiştir. Çalıştığı süre boyunca kaydedilen ağ trafiği üzerinde yapılan analizde ise herhangi bir zararlı ağ trafiği tespiti yapılmamıştır.

Docker Hub üzerinde "covid29" kullanıcısına ait "covid29/vaccine4all" kapsayıcısının dinamik analiz kapsamında yapılan port taramasında TCP/80 portunun açık olduğu ve bu port üzerinde "Apache/2.4.10 (Debian) PHP/5.6.10" servisi çalıştığı tespit edilmiştir. Kapsayıcıya ait kaydedilen ağ trafiğinin analizinde de zararlı ağ trafiği tespiti yapılmamıştır. Bu kullanıcıya ait "covid29/commandcontrol" kapsayıcısının dinamik analizindeyse açık port tespiti yapılmamış olup, kaydedilen ağ trafiğinin analizindeyse Snort3 Kara liste Ip adreslerine ICMP protokolü aracılığıyla erişilmeye çalıştığı model tarafından tespit edilmiştir.

Kullanıcılara bir anti virüs hizmeti veriyor gibi paylaşılan ve "Js.Coinminer.Generic-6836639" zararlı yazılımını barındıran "antiviruss/scanner" kapsayıcısının model tarafından dinamik analiz kapsamında yapılan port taramasında, TCP/80 portunun açık olduğu ve bu port üzerinde "Apache/2.4.10 (Debian) PHP/5.6.10" servisi çalıştığı tespit edilmiştir.

6. SONUÇ

Siber saldırıların her geçen gün artması, saldırı vektörlerinin karmaşıklaşması ve saldırgan motivasyonunun yüksekliği gibi etkenler, tasarım aşamasında güvenlik prensibinin önemini ortaya koymaktadır. Güvenlik kontrolleri yapılmadan kullanılan görüntüler, hizmet dışı bırakma, yetki yükseltme, kripto madenciliği, zararlı yazılım enjeksiyonu ve konak işletim sisteminin botnet ağlarının bir parçası haline gelmesi gibi istenmeyen durumlara sebep olabilir.

Docker Hub üzerinde bireysel kullanıcılar tarafından paylaşılan görüntülerin, binlerce kez indirilmesi görüntünün güvenli olduğu anlamını taşımamaktadır. Belirli amaçlar için oluşturulup Docker Hub repolarına yüklenen bu görüntülerin güncelleme yapılmadan yıllarca bekliyor olması siber saldırı tehlikesini arttırmaktadır. Görüntünün zararlı yazılım barındırdığının tespitine kadar geçen sürede binlerce kullanıcı etkilenebilir. Önerilen model, mikro servis mimarisi yaklaşımı ile bulut bilişim üzerinde hizmet veren veya hizmet vermeyi planlayan kurumsal ve bireysel kullanıcılara, Docker görüntü seçiminde güvenlik raporu sunmaktadır. Güvenli Docker görüntüsünün kullanımıyla kurumların ve bireylerin uğrayacağı maddi zararların yanı sıra itibar kayıplarının da önüne geçilebilecektir. Modelde, Docker Hub üzerinde yüz binden fazla indirilen ve Docker kapsayıcıların çalışırlığının kontrolünü yaptığı düşünülen “docheck/health” görüntüsü ve aynı kullanıcı tarafından paylaşılan “docheck/ax” görüntüsü aslında kripto madenciliği yapmaktadır. Kripto madenciliği yapan bu görüntüler başarılı bir şekilde model tarafından tespit edilebilmiştir. Ayrıca içinde Avrupa Bilgisayar Anti virüs Araştırma Enstitüsü tarafından oluşturulan test dosyası bulunan “buamod/eicar” görüntüsü de model tarafından yapılan statik analiz ile tespit edilmiştir.

Docker Hub üzerinde paylaşılan “ubvntu/utnubu”, “ubvntu/vbuntu” ve “tempsbro/tempsbro” görüntüleri üzerinde de kripto madenciliği yapan zararlı yazılım tespiti yapılmıştır.

Kullanıcılara bir anti virüs hizmeti veriyor gibi isimlendirilerek paylaşılan “antiviruss/scanner” görüntüsü üzerinde de zararlı yazılım ve açık port tespiti yapılmıştır. Güncel konularla ilgili bilgi paylaşımı yapıyor izlenimi verilerek oluşturulan Docker görüntüleri, çalıştırıldıkları sistemler üzerinde istenmeyen sonuçlara neden olabilir. Bu şekilde hazırlanan “covid29/vaccine4all”, “covid29/commandcontrol” ile” tester122/cehdumpsv11” görüntüleri üzerinde zararlı yazılım tespitinin yanı sıra açık portların da bulunduğu, ayrıca zararlı ağ trafiği oluşturdukları tespit edilmiştir.

Docker görüntüleri üzerinde yapılan statik analiz sonucunda zafiyetli paket veya kötücül yazılım tespit edilemeyebilir. Statik analizde görüntüde zafiyetli paket veya kötücül yazılım tespit edilmemesi, görüntü güvenliğini garanti etmemektedir. Örneğin “osengul/maliciousdocker” görüntüsünün yapılan statik analizinde zafiyetli paket ve kötücül yazılım tespit edilmemiştir. Ancak AlphaSoc firması tarafından siber güvenlik operasyon merkezlerinde görevli personelin eğitimini desteklemek amacıyla geliştirilen “Network Flight Simulator” aracıyla zararlı olduğu bilinen alan adlarına doğru ağ trafiği oluşturulmakta ve zararlı ağ trafiği taklit edilmektedir. Statik analizinde zafiyetli paket ve kötücül yazılım tespit edilmeyen bu görüntünün yalıtılmış bir kum havuzunda belli bir süre çalıştırılması sonucu kaydedilen ağ trafiğinin Snort3 analizinde zararlı ağ trafiği olarak değerlendirilebilecek DNS sorguları gerçekleştirdiği tespit edilmiştir.

Bayrak yakalama yarışmasında kullanılmak üzere oluşturulan “osengul/flag” görüntüsü üzerinde varsayılan kullanıcı adı ve şifreyle bağlanılabilen TCP 22 ssh portunun açık olduğu ve msfvenom aracıyla oluşturularak görüntü içerisine yerleştirilen zararlı yazılım model tarafından başarılı bir şekilde tespit edilmiştir.

Modelde statik analiz aşamasında Anchore aracına ait servislerin hazır olması, CVE veritabanınının güncellemesi ve incelenecek görüntünün boyutuna göre analiz süresi değişkenlik göstermektedir. Gelişmiş zararlı yazılımlar, kum havuzunda olduğunun, çalıştırıldığı ortamın ve güvenlik tedbirlerinin tespitini yapacak şekilde kodlanmaktadır. Bu tür zararlı yazılım barındıran görüntüler, dinamik analizde tespit edilemeyebilir. Görüntülerin içerisine gizlenen bir takım kötücül yazılımlar, çalışmak için kapsayıcıların belli bir süre konak işletim sistemi üzerinde hizmet vermesini beklemektedirler. Bu şekilde çalışması için belli şartların oluşmasını bekleyen kötücül yazılımların görüntü içerisinde tespiti gittikçe zorlaşmaktadır

Ayrıca Docker görüntü güvenliğine yönelik yapılan dinamik analizde, özel parametre alarak çalışan görüntülere dikkat edilmelidir. Örneğin seçilen postgresql görüntüsü özel parametre girilmeden çalıştırılamamış ve dinamik analizi yapılamamıştır. Bazı Docker görüntüleri sundukları hizmetin gereği olarak kullanıcıdan bir takım parametreler alarak çalışmaları gerekmektedir. Bu tür görüntüler parametre almadan çalışmamaktadırlar. Dolayısıyla modelde bu tür görüntülerin yalnızca statik analizleri gerçekleştirilmiştir.

Yapılacak çalışmalarda, kapsayıcı ağ trafiğinin saldırı tespit/önleme sistemleri ile sürekli izlenmesinin ve makine öğrenmesi ile davranışsal anomali tespitinin esas alınması Docker görüntü güvenliğine katkı sağlayacağı değerlendirilmektedir.

KAYNAKLAR

[1] **Javed, O.** and **Toor, S.** (2021) Understanding the Quality of Container Security Vulnerability Detection Tools. 1–11.

[2] **Burniske, C.** (2014) "Containers: The Next Generation of Virtualization? | ARK" https://ark-invest.com/articles/analyst-research/containers-virtualization/ Alındığı Tarih: 23.10.2021.

[3] **Internet:** "What Are Containers?" https://rancher.com/learning-paths/what-are-containers/ Alındığı Tarih: 02.05.2021.

[4] **Wist, K.**, **Helsem, M.**, and **Gligoroski, D.** (2020) Vulnerability analysis of 2500 docker hub images. *ArXiv*. (June),.

[5] "Introducing the Docker Index: Insight from the World's Most Popular Container Registry - Docker Blog" https://www.docker.com/blog/introducing-the-docker-index/ Alındığı Tarih: 02.05.2021.

[6] **Huang, D.**, **Cui, H.**, **Wen, S.**, and **Huang, C.** (2019) Security Analysis and Threats Detection Techniques on Docker Container. *2019 IEEE 5th International Conference on Computer and Communications, ICCC 2019*. 1214–1220.

[7] **Loukidis-Andreou, F.**, **Giannakopoulos, I.**, **Doka, K.**, and **Koziris, N.** (2018) Docker-Sec: A fully automated container security enhancement mechanism. *Proceedings - International Conference on Distributed Computing Systems*. 2018-July 1561–1564.

[8] **Brady, K.**, **Moon, S.**, **Nguyen, T.**, and **Coffman, J.** (2020) Docker Container Security in Cloud Computing. *2020 10th Annual Computing and Communication Workshop and Conference, CCWC 2020*. 975–980.

[9] **Mullinix, S.P.**, **Konomi, E.**, **Townsend, R.D.**, and **Parizi, R.M.** (2020) On security measures for containerized applications imaged with docker. *ArXiv*.

[10] **Bashari Rad, B.**, **John Bhatti, H.**, and **Ahmadi, M.** (2017) An Introduction to Docker and Analysis of its Performance. *IJCSNS International Journal of Computer Science and Network Security*. 17 (3), 228–235.

[11] **Sharma, P.**, **Chaufournier, L.**, **Shenoy, P.**, and **Tay, Y.C.** (2016) Containers and virtual machines at scale: A comparative study. in: Proc. 17th Int. Middlew. Conf. Middlew. 2016, Association for Computing Machinery, Inc, .

[12] **Shu, R.**, **Gu, X.**, and **Enck, W.** (2017) A study of security vulnerabilities on docker hub. *CODASPY 2017 - Proceedings of the 7th ACM Conference on Data and Application Security and Privacy*. 269–280.

[13] **Rad, B.B.**, **Bhatti, H.J.**, and **Ahmadi, M.** (2017) An Introduction to Docker and Analysis of its Performance. 17 (3), 228–235.

[14] **Tomar, A.**, **Jeena, D.**, **Mishra, P.**, and **Bisht, R.** (2020) Docker security: A threat model, attack taxonomy and real-time attack scenario of DoS. *Proceedings of the Confluence 2020 - 10th International Conference on Cloud Computing, Data Science and Engineering*. 150–155.

[15] **Liu, P., Ji, S., Fu, L., Lu, K., Zhang, X., Lee, W.H., et al.** (2020) Understanding the security risks of docker hub. in: Lect. Notes Comput. Sci. (Including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics), Springer Science and Business Media Deutschland GmbH, pp. 257–276.

[16] **Wenhao, J.** and **Zheng, L.** (2020) Vulnerability Analysis and Security Research of Docker Container. *Proceedings of 2020 IEEE 3rd International Conference on Information Systems and Computer Aided Education, ICISCAE 2020*. 354–357.

[17] **Kwon, S.** and **Lee, J.H.** (2020) DIVDS: Docker Image Vulnerability Diagnostic System. *IEEE Access*. 8 42666–42673.

[18] **Zerouali, A.**, **Mens, T.**, **Robles, G.**, and **Gonzalez-Barahona, J.M.** (2018) On the relation between outdated docker containers, severity vulnerabilities and bugs. *ArXiv*. 491–501.

[19] **Randal, A.** The Ideal Versus the Real: Revisiting the History of Virtual Machines and Containers. .

[20] **Marinescu, D.** and **Kröger, R.** (2007) State of the art in autonomic computing and virtualization. .

[21] **Eder Betreuer, M.**, **Kinkelin, H.**, and **Netzarchitekturen, L.** Hypervisor-vs. Container-based Virtualization.

[22] **Yadav, A.K.**, **Garg, M.L.**, and **Ritika** (2019) Docker containers versus virtual machine-based virtualization. Springer Singapore, .

[23] **CANSU GÖKHAN** (2017) SCALABILITY ISSUES IN BIG DATA ON CLOUD: A COMPARISON OF VIRTUAL MACHINES AND LINUX CONTAINERS. (January), 1–14.

[24] **DOĞRU, A.** (2019) Sunucu Sanallaştırma ve Uygulama Karşılaştırması, Maltepe Üniversitesi, Fen Bilimleri Enstitüsü, İstanbul, 538721, 2019.

[25] **Internet:** "Containers Red Hat Enterprise Linux Atomic Host 7" https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html/overview_of_containers_in_red_hat_systems/introduction_to_linux_containers Alındığı Tarih: 23.10.2021.

[26] **Hayden, M.** and **Carbone, R.** Securing Linux Containers GIAC (GCUX) Gold Certification. .

[27] **Martin, A.**, **Raponi, S.**, **Combe, T.**, and **Di Pietro, R.** (2018) Docker ecosystem – Vulnerability Analysis. *Computer Communications*. 122 30–43.

[28] **Yasrab, R.** and **Technology, I.** Mitigating Docker Security Issues.

[29] **Internet:** "Market Share Containerization Docker" https://www.datanyze.com/market-share/containerization--321/docker-market-share Alındığı Tarih: 09.09.2021.

[30] **wikipedia** (2020) "Docker Software"

https://en.wikipedia.org/wiki/Docker_(software) Alındığı Tarih: 24.10.2021.

[31] **Zerouali, A.** (2018) Analyzing technical lag in docker images. *CEUR Workshop Proceedings*. 2361 6–10.

[32] **Pittenger, M.** (2016) Addressing the security challenges of using containers. *Network Security*. 2016 (12), 5–8.

[33] **Sumeet Gyanchandani** "Docker Storage" https://towardsdatascience.com/docker-storage-598e385f4efe Alındığı Tarih: 24.10.2021.

[34] **Zeng, H.**, **Wang, B.**, **Deng, W.**, and **Zhang, W.** (2017) Measurement and evaluation for docker container networking. *Proceedings - 2017 International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery, CyberC 2017*. 2018-Janua 105–108.

[35] **Russ McKendrick** (2020) Mastering Docker - Fourth Edition. .

[36] **Fabrizio Soppelsa** and **Chanwit Kaewkasi** (2016) Native Docker Clustering with Swarm. .

[37] **Soltesz, S.**, **Pötzl, H.**, **Fiuczynski, M.E.**, **Bavier, A.**, and **Peterson, L.** (2012) Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors Stephen. *Aviation Week and Space Technology*. 174 (19), 275–287.

[38] **Internet:** (2019) "Anchore" https://github.com/anchore/anchore-engine Alındığı Tarih: 24.10.2021.

[39] **Valance, J.** "Use Anchore Policies to Reach CIS Docker Benchmark" https://anchore.com/blog/cis-docker-benchmark/ Alındığı Tarih: 24.10.2021.

[40] **Internet:** "Anchore Engine Overview" https://engine.anchore.io/docs/general/ Alındığı Tarih: 24.10.2021.

[41] **Cisco Systems, I.** (2021) "Clamav Documentation" https://docs.clamav.net/ Alındığı Tarih: 24.10.2021.

[42] **Acharjee, D.** "Using ClamAV to Detect and Prevent Malware" https://www.opensourceforu.com/2021/07/using-clamav-to-detect-and-prevent-malware/ Alındığı Tarih: 24.10.2021.

[43] **Internet:** "20 Million Miners: Finding Malicious Cryptojacking Images in Docker Hub" https://unit42.paloaltonetworks.com/malicious-cryptojacking-images/ Alındığı Tarih: 09.09.2021.

[44] **Internet:** "Docker Image | Docker Hub" https://hub.docker.com/r/imiell/bad-dockerfile Alındığı Tarih: 09.09.2021.

EKLER

EK 1: Anchore statik analiz aracı docker-compose dosya içeriği

EK 2: ClamAV Dockerfile dosya içeriği

EK 3: Snort3 Dockerfile dosya içeriği

EK 1

```
version: '2.1'
volumes:
  anchore-db-volume:
    external: false
services:
  api:
    image: anchore/anchore-engine:v0.10.0
    depends_on:
      - db
      - catalog
    ports:
      - "8228:8228"
    logging:
      driver: "json-file"
      options:
        max-size: 100m
    environment:
      - ANCHORE_ENDPOINT_HOSTNAME=api
      - ANCHORE_ADMIN_PASSWORD=foobar
      - ANCHORE_DB_HOST=db
      - ANCHORE_DB_PASSWORD=mysecretpassword
    command: ["anchore-manager", "service", "start", "apiext"]
  catalog:
    image: anchore/anchore-engine:v0.10.0
    depends_on:
      - db
    logging:
      driver: "json-file"
      options:
        max-size: 100m

    expose:
      - 8228
    environment:
      - ANCHORE_ENDPOINT_HOSTNAME=catalog
      - ANCHORE_ADMIN_PASSWORD=foobar
      - ANCHORE_DB_HOST=db
      - ANCHORE_DB_PASSWORD=mysecretpassword
    command: ["anchore-manager", "service", "start", "catalog"]
  queue:
    image: anchore/anchore-engine:v0.10.0
    depends_on:
      - db
      - catalog
    expose:
      - 8228
    logging:
      driver: "json-file"
```

```
    options:
      max-size: 100m
  environment:
    - ANCHORE_ENDPOINT_HOSTNAME=queue
    - ANCHORE_ADMIN_PASSWORD=foobar
    - ANCHORE_DB_HOST=db
    - ANCHORE_DB_PASSWORD=mysecretpassword
  command: ["anchore-manager", "service", "start", "simplequeue"]
policy-engine:
  image: anchore/anchore-engine:v0.10.0
  depends_on:
    - db
    - catalog
  expose:
    - 8228
  logging:
    driver: "json-file"
    options:
      max-size: 100m
  environment:
    - ANCHORE_ENDPOINT_HOSTNAME=policy-engine
    - ANCHORE_ADMIN_PASSWORD=foobar
    - ANCHORE_DB_HOST=db
    - ANCHORE_DB_PASSWORD=mysecretpassword
  command: ["anchore-manager", "service", "start", "policy_engine"]
analyzer:
  image: anchore/anchore-engine:v0.10.0
  depends_on:
    - db
    - catalog
  expose:
    - 8228
  logging:
    driver: "json-file"
    options:
      max-size: 100m
  environment:
    - ANCHORE_ENDPOINT_HOSTNAME=analyzer
    - ANCHORE_ADMIN_PASSWORD=foobar
    - ANCHORE_DB_HOST=db
    - ANCHORE_DB_PASSWORD=mysecretpassword
  volumes:
    - /analysis_scratch
  command: ["anchore-manager", "service", "start", "analyzer"]
db:
  image: "postgres:9"
  volumes:
    - anchore-db-volume:/var/lib/postgresql/data
  environment:
    - POSTGRES_PASSWORD=mysecretpassword
```

```
expose:
  - 5432
logging:
  driver: "json-file"
  options:
    max-size: 100m
healthcheck:
  test: ["CMD-SHELL", "pg_isready -U postgres"]
```

EK 2

```
FROM ubuntu:20.04
MAINTAINER Ozkan SENGUL

USER root

# copy assets to image
COPY ./assets /usr/local

# install antivirus and dependencies, get the latest clamav and maldet signatures
RUN apt-get -y update && \
    apt-get -y upgrade && \
    apt-get install -y apt-utils clamav clamav-daemon curl inotify-tools supervisor host
tar wget systemctl && \
    mkdir -p /var/log/supervisor && \
    mkdir -p /var/log/cron && \
    cd /usr/local/ && chmod +x *.sh && sync && \
    cd /usr/local/bin && chmod +x *.sh && sync && \
    /usr/local/install_maldet.sh && \
    /usr/local/install_antivirus.sh && \
    apt-get -y remove curl apt-utils && \
    rm -rf /var/cache/* && \
    freshclam && \
    maldet -u -d

# export volumes (uncomment if you do not mount these volumes at runtime or via
docker-compose)
# VOLUME /data/av/queue
# VOLUME /data/av/ok
# VOLUME /data/av/nok

ENTRYPOINT ["/usr/local/entrypoint.sh"]
```

EK 3

```
FROM ubuntu:latest

RUN apt-get update && DEBIAN_FRONTEND=noninteractive apt-get install -yq \
    wget \
    build-essential \
    libpcap-dev \
    libpcre3-dev \
    libnet1-dev \
    zlib1g-dev \
    luajit \
    hwloc \
    libdnet-dev \
    libdumbnet-dev \
    bison \
    flex \
    liblzma-dev \
    openssl \
    libssl-dev \
    pkg-config \
    libhwloc-dev \
    cmake \
    cpputest \
    libsqlite3-dev \
    uuid-dev \
    libcmocka-dev \
    libnetfilter-queue-dev \
    libmnl-dev \
    autotools-dev \
    libluajit-5.1-dev \
    libunwind-dev \
    iproute2 \
    net-tools \
    sudo \
    ethtool \
    libtool \
    git \
    autoconf \
    ragel \
    libboost-dev \
    libboost-all-dev \
    systemd \
    libcrypt-ssleay-perl \
    liblwp-useragent-determined-perl \
    && apt-get clean && rm -rf /var/cache/apt/*
```

```
WORKDIR /opt
ENV SAFEC_VERSION 02092020
RUN wget
https://github.com/rurban/safeclib/releases/download/v02092020/libsafec-
${SAFEC_VERSION}.tar.gz \
   && tar xvfz libsafec-${SAFEC_VERSION}.tar.gz \
   && cd libsafec-${SAFEC_VERSION}.0-g6d921f \
   && ./configure \
   && make \
   && sudo make install

ENV PCRE_VERSION 10.37
RUN wget https://ftp.pcre.org/pub/pcre/pcre2-${PCRE_VERSION}.tar.gz \
   && tar xzvf pcre2-${PCRE_VERSION}.tar.gz \
   && cd pcre2-${PCRE_VERSION} \
   && ./configure && make && sudo make install

ENV GP_TOOLS_VERSION 2.9.1
RUN wget https://github.com/gperftools/gperftools/releases/download/gperftools-
${GP_TOOLS_VERSION}/gperftools-${GP_TOOLS_VERSION}.tar.gz \
   && tar xzvf gperftools-${GP_TOOLS_VERSION}.tar.gz \
   && cd gperftools-${GP_TOOLS_VERSION} \
   && ./configure && make && sudo make install

ENV HYPERSCAN_VESRSION 5.4.0-2
RUN wget
https://launchpad.net/ubuntu/+archive/primary/+sourcefiles/hyperscan/5.4.0-
2/hyperscan_5.4.0.orig.tar.gz \
   && tar xvzf hyperscan_5.4.0.orig.tar.gz \
   && mkdir hyperscan-${HYPERSCAN_VESRSION}-build \
   && cd hyperscan-${HYPERSCAN_VESRSION}-build \
   && cmake -DCMAKE_INSTALL_PREFIX=/usr/local ../hyperscan-5.4.0 \
   && make && sudo make install

ENV FLATBUFFERS_VESRSION 2.0.0
RUN wget
https://github.com/google/flatbuffers/archive/refs/tags/v${FLATBUFFERS_VESRSI
ON}.tar.gz -O flatbuffers-v${FLATBUFFERS_VESRSION}.tar.gz \
   && tar xvzf flatbuffers-v${FLATBUFFERS_VESRSION}.tar.gz \
   && mkdir flatbuffers-build \
   && cd flatbuffers-build \
   && cmake ../flatbuffers-${FLATBUFFERS_VESRSION} \
   && make && sudo make install

# DAQ
ENV DAQ_VERSION 3.0.3
RUN wget
https://github.com/snort3/libdaq/archive/refs/tags/v${DAQ_VERSION}.tar.gz \
   && tar xvfz v${DAQ_VERSION}.tar.gz \
   && cd libdaq-${DAQ_VERSION} \
```

```
    && ./bootstrap \
    && ./configure \
    && make \
    && make install

RUN ldconfig
# Snort 3.1.0
ENV MY_PATH=/usr/local/snort
ENV SNORT_VERSION 3.1.5.0
RUN wget
https://github.com/snort3/snort3/archive/refs/tags/${SNORT_VERSION}.tar.gz \
    && tar xvfz ${SNORT_VERSION}.tar.gz \
    && cd snort3-${SNORT_VERSION} \
    && ./configure_cmake.sh --prefix=${MY_PATH} \
    && cd build \
    && make -j $(nproc) install

RUN ldconfig

# OpenAppID - Device detection
ENV OPEN_APP_ID 17843
RUN wget https://www.snort.org/downloads/openappid/${OPEN_APP_ID}  -O
OpenAppId-${OPEN_APP_ID}.tgz \
    && tar xvfz OpenAppId-${OPEN_APP_ID}.tgz \
    && cp -R odp /usr/local/lib/

ENV SNORT_RULES_SNAPSHOT 3190
COPY snortrules-snapshot-${SNORT_RULES_SNAPSHOT} /opt/
COPY *.sh /opt/

RUN mkdir -p /var/log/snort && \
    mkdir -p /usr/local/lib/snort_dynamicrules && \
    mkdir -p /etc/snort && \
    mkdir -p /etc/snort/rules && \
    mkdir -p /etc/snort/rules/iplists && \
    mkdir -p /etc/snort/preproc_rules && \
    mkdir -p /etc/snort/etc && \
    cp -r /opt/rules /etc/snort && \
    cp -r /opt/so_rules /etc/snort && \
    cp -r /opt/etc /etc/snort && \
    cp -r /opt/builtins /etc/snort && \
    cp -r /opt/rules/iplists /etc/snort/rules && \
    # Custom rules goes to local.rules
    # Will be copied an external file to Docker
    # COPY local.rules /etc/snort/rules/local.rules
    touch /etc/snort/rules/local.rules && \
    touch /etc/snort/rules/iplists/white_list.rules && \
    touch /etc/snort/rules/iplists/black_list.rules
```

```
# Pulledpork
RUN wget https://github.com/shirkdog/pulledpork/archive/master.tar.gz -O
pulledpork-master.tar.gz \
   && tar xzvf pulledpork-master.tar.gz \
   && cd pulledpork-master \
   && cp pulledpork.pl /usr/bin/ \
   && chmod 755 /usr/bin/pulledpork.pl \
   && cp etc/* /etc/snort/ \
   && cpan install LWP::Protocol::https \
   && cpan install Crypt::SSLeay  \
   && cpan Mozilla::CA IO::Socket::SSL

# Check Pulledpork was installed
RUN /usr/bin/pulledpork.pl -V && sleep 15

# Pulledpork conf
COPY pulledpork.conf /etc/snort/pulledpork.conf
COPY disablesid.conf /etc/snort/disablesid.conf

# COPY local rules across
COPY /rules/local.rules /etc/snort/rules/local.rules

RUN wget -O /etc/snort/rules/iplists/default.blocklist
https://www.snort.org/downloads/ip-block-list
# Clean up APT when done.
RUN apt-get clean && rm -rf /var/lib/apt/lists/* /tmp/* /var/tmp/* \
   /opt/${SNORT_VERSION}.tar.gz /opt/v${DAQ_VERSION}.tar.gz

ENV INTERFACE 'eth0'
ENV LUA_PATH=${MY_PATH}/include/snort/lua/\?.lua\;\;
ENV SNORT_LUA_PATH=${MY_PATH}/etc/snort
ENV PATH="/usr/local/snort/bin:$PATH"
ENV PCAP test
ENV REPORT report

# To check what that dir contains
# RUN cd /usr/local/snort/bin/ && ls -la && sleep 30

# Network interface service --> Not working
# RUN ls -la /lib/systemd/system/ && sleep 30
# COPY ethtool.service /lib/systemd/system/
# RUN sudo service enable --now ethtool \
#     && sudo service ethtool start

# HOME_NET config --> chage this with the right IP adresses where snort should
monitoring
#ARG
SNORT_HOME_NET="192.168.0.0/16,172.16.0.0/12,0.0.0.0/8,172.17.0.0/16"
#RUN sed -i "s#^HOME_NET =.*#HOME_NET = '$SNORT_HOME_NET'#"
/etc/snort/etc/snort.lua
```

```
COPY snortson.lua /etc/snort/etc/snort.lua
# Validate an installation
#RUN ${MY_PATH}/bin/snort -c /etc/snort/etc/snort.lua
RUN chmod a+x /opt/*

# Add the script that allows the rules to be updated when the container is running
ARG PPORK_OINKCODE “OINK KODUNU BURAYA GİRMELİSİNİZ”

RUN if [ ! -z $PPORK_OINKCODE ]; then  bash update-rules.sh
"$PPORK_OINKCODE"; fi

# Exposed port
#EXPOSE 8080
# Let's run snort!
CMD ["/bin/bash", "/opt/entrypoint.sh"]
#CMD ["snort"]
#CMD ["-i", "eth0", "-A", "fast", "-s", "65535", "-k", "none"]
#ENTRYPOINT ["/bin/sh /opt/entrypoint.sh"]
#CMD ["sleep 120"]
# CMD ["/bin/bash", "-c", "pulledpork.pl -c /etc/snort/pulledpork.conf -l -P -E -H
SIGHUP"]
# CMD ["/bin/bash", "-c", "snort -c /etc/snort/etc/snort.lua -r /tmp/test.pcap -q -k
none -A fast > report.txt 2>&1"]
#CMD snort -c /etc/snort/etc/snort.lua -r /tmp/${PCAP} -q -k none -A fast >
${REPORT}
#CMD snort -c /etc/snort/etc/snort.lua -r /tmp/${PCAP} -q -k none -A fast >
${REPORT}
```