容器镜像存储原理及其安全风险研究

Research of Docker Image Storage Principle and Security Risk

丁　攀,徐　雷,刘　安,苏俐竹(中国联通研究院,北京 100048)
Ding Pan,Xu Lei,Liu An,Su Lizhu(China Unicom Research Institute,Beijing 100048,China)

摘　要:
随着云原生技术的发展,容器应用越来越广泛。作为容器运行的载体,镜像对于容器安全起着至关重要的作用。为更好理解容器镜像安全扫描的原理,指导容器安全实践,详细分析镜像在本地和仓库中的存储原理,总结其存储规律。最后,以1个含有木马病毒的镜像为例,分析镜像非法篡改过程。

关键词:
容器镜像,镜像存储,镜像扫描
doi:10.12045/j.issn.1007-3043.2022.09.017
文章编号:1007-3043(2022)09-0082-06
中图分类号:TN915.08
文献标识码:A
开放科学(资源服务)标识码(OSID):

Abstract:
With the development of cloud-native technologies, containers are used more and more widely. As the carrier of container operation, image plays a crucial role in the security of container. In order to better understand the principle of image security scanning, and guide container security practices, it fully analyzes the storage principle of local images and registry images, and summarizes the storage rules. Finally, a Trojan image is taken as an example to analyze the process of image illegal tampering.

Keywords:
Docker image; Image storage; Image scanning

引用格式:丁攀,徐雷,刘安,等. 容器镜像存储原理及其安全风险研究[J]. 邮电设计技术,2022(9):82-87.

1 概述

近10年来,云计算的发展取得了显著的成就,虚拟化作为其中关键技术发展日益迅速,特别是轻量级的虚拟化技术——Docker容器技术。Docker提供一种可移植、可重用且自动化的方式来打包和运行应用。随着容器技术的广泛应用,Docker容器的安全问题日益突出,尤其是容器镜像安全问题。

镜像仓库是容器镜像的主要传播方式,Docker Hub作为Docker官方的镜像仓库,对于镜像上传过程缺乏完善的监督,导致镜像的版本、质量和安全性处于不可控的状态。文献[1]分析Docker Hub仓库中的2 500个镜像后指出,高达82%被认证的镜像包含至少1个高危漏洞,而不含漏洞的镜像仅占17.8%。可见镜像安全已经成为容器面临的最主要风险之一。为方便读者更好理解镜像的脆弱性,本文详细分析了镜像在仓库和本地存储的结构特点,并针对性地分析了镜像存在的安全风险。最后以1个含有木马病毒的镜像为例,通过对比镜像文件的变化,分析了非法篡改镜像的过程,最后简要介绍了镜像漏洞扫描器的工作原理。

2 容器镜像及其特点

Docker镜像是容器运行的模板,它含有启动

收稿日期:2022-07-15

Docker容器所需要的文件系统及其内容,Docker镜像文件与Docker容器的配置文件共同构成了Docker容器的静态文件系统运行环境RootFS,镜像是容器的静态视角,而容器则是镜像的运行状态。容器镜像主要具有以下特点。

a)分层存储:Docker镜像采用分层存储的方式,每个镜像都由一系列的层文件(Layer)组成,这些文件按照一定顺序排列,不同镜像层可以共享底层镜像层文件,从而达到节省存储空间的目的。

b)写时复制:在未更改镜像文件时,所有的容器共享同样的镜像文件,当需要修改容器内容时,只需修改最上层的镜像文件。修改后的文件存储在容器的读写层,写时复制与分层存储一样,节省了存储空间。

c)内容寻址:内容寻址是指系统根据镜像内容的Hash值来索引镜像位置,在对镜像执行pull、push、load和save等操作时,可以通过Hash值验证镜像数据的完整性,内容寻址既提高了镜像的安全性,又降低了镜像名称冲突的可能。

d)联合挂载:Docker镜像通过联合挂载技术实现多层文件的叠加,如AUFS、OverlayFS等。OverlayFS将Linux主机上的2个目录合并成1个目录,对外提供一个统一的视角,如图1所示。下层目录为只读的镜像层。上层目录为可写的容器层。合并对外展示的统一视图称为merged层,在合并后的视图中,上层目录会覆盖下层目录的内容。当需要修改一个文件时,通过写时复制技术将文件从只读的lowerdir复制到可写的upperdir,进行修改后保存在upperdir层。

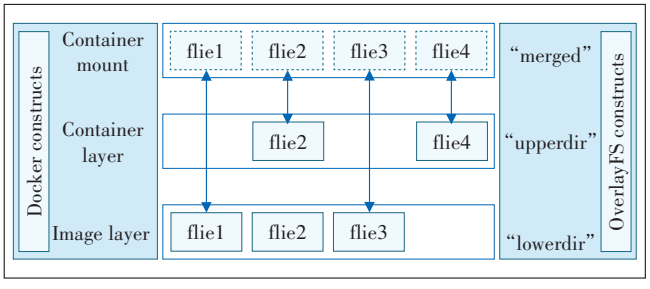

图1　联合文件系统OverlayFS架构图

3 容器镜像存储与安全风险

3.1 镜像在仓库中存储与风险

为了便于理解,本文以Registry镜像仓库为例,来详细介绍镜像在仓库中的存储架构,仓库以容器的形式启动,镜像在容器中默认的存储位置是/var/lib/registry,通过-v命令将该目录挂载到本地/var/lib/registry,该目录包括blobs和repositories 2个目录,其中blobs目录存储镜像文件,repositories目录存储镜像元数据,镜像元数据与镜像文件被设计成独立隔离存储。以ubuntu:16.04镜像为例,当registry中存储该镜像后,目录blobs和repositories存储的具体内容如图2所示。

manifest文件是存储在registry中作为Docker镜像的元数据文件,在镜像pull、push、save和load中作为镜像结构和基础信息描述文件,当镜像被pull到宿主机时,manifest被转化为本地镜像的配置文件。在存储镜像元数据文件的目录repositories中,_layers目录下link文本文件指向blobs目录下与之对应的data文件,而_manifests目录包含镜像revisions和tags信息,其中的current/link文件链接到镜像的manifest文件。

blobs目录存储镜像文件,它是以data为名的压缩文件,目录名称为data的sha256数值,即在上文中所述的内容寻址,结果为64位16进制字符串。为了便于展示,此处缩减为12位字符串,即58690f9b18fc、b51569e7c507、da8ef40b9eca和fb15d46c38dc,也被称为LayerID。通过解压缩可以看到data存储的详细内容。Docker Daemon在拉取镜像时,会下载data文件并将解压缩内容保存在本地。

目录b6f507652425存储的是镜像配置文件信息,包括操作系统、镜像层Diff-ID、镜像创建历史、环境变量、执行命令等信息。目录a3785f78ab85存储的是镜像的manifest,正如文件_manifests/tags/16.04/current/link所链接的,manifest记录了镜像所包含的layer信息,其中主要信息及其含义分别如下。

mediaType:表示镜像层文件的类型及其压缩格式。

size:表示该压缩包文件的大小。

digest:表示该压缩包文件的Hash值。

通过读取manifest信息,可以获取镜像层及各层所占空间等信息,这些信息和图2所反馈的信息是一致的。目录b6f5076524258存储的是镜像配置相关信息,包括镜像架构、创建过程、容器配置等内容,与docker inspect获取的信息基本一致。除了上述信息以外,还可以验证同一镜像在不同镜像仓库中的manifest存储路径和内容、blobs目录下的存储路径和内容全都是一致的。

本地镜像在分层存储过程中,镜像仓库中存储的

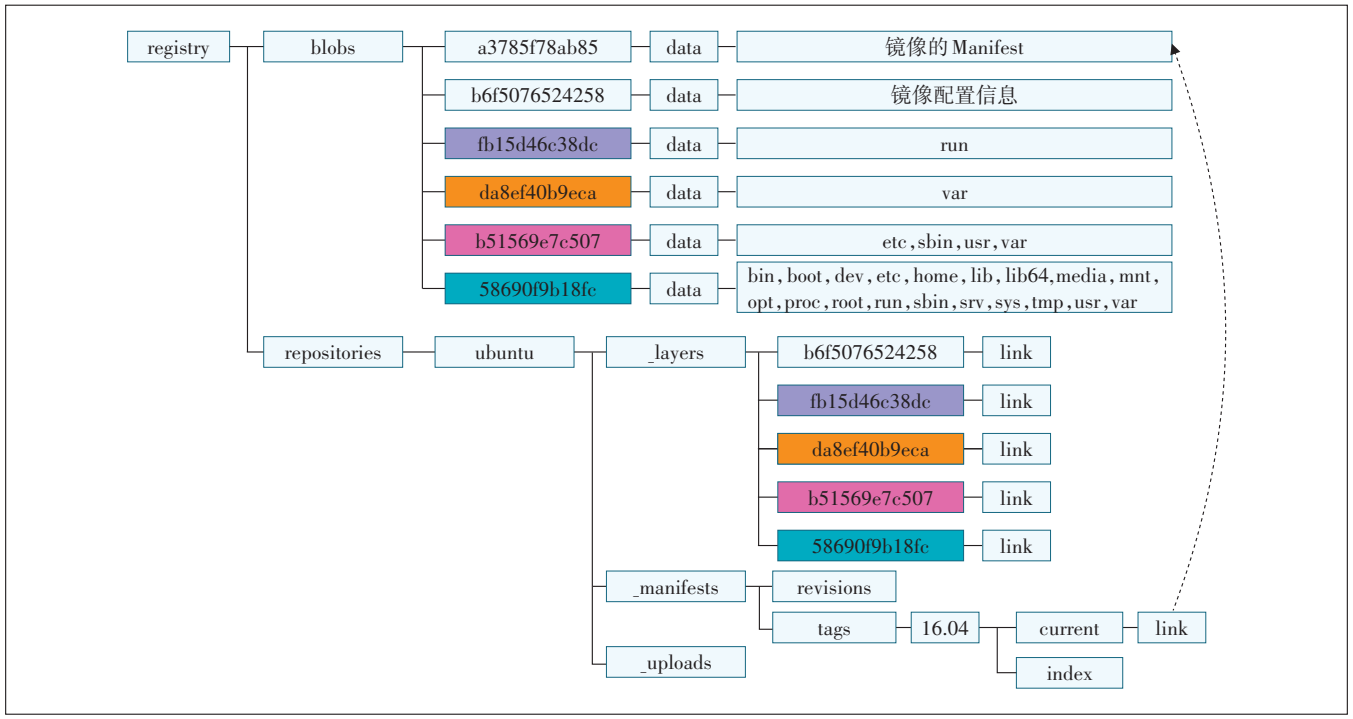

图2　镜像 ubuntu：16.04 在仓库中存储

镜像可能包含漏洞或木马病毒文件，当被终端下载且未被检查时，就有可能导致危险容器镜像的泛滥。除此之外，容器镜像也可能包含数据库密码、证书密钥以及敏感环境变量等，存在信息泄露的风险。

3.2 镜像传输过程与风险

docker pull 命令可以下载镜像，通过下载的提示信息，可以看到镜像 ubuntu：16.04 所对应镜像层文件（58690f9b18fc、b51569e7c507、da8ef40b9eca、fb15d46c38dc）都被下载保存到本地，这些层文件 ID 即上节所述的 LayerID。从镜像仓库中下载镜像层文件的过程如图 3 所示。

镜像下载的第 1 步是获取镜像的 manifest 文件，成功获取 manifest 之后，客户端通过 digest 来独立下载镜像的层文件。如步骤 5 所示，此处请求下载镜像的格式为 tar.gz，而 Docker Daemon 在实际的操作过程中，会将下载容器镜像进行解压缩并保存在本地，而原压缩文件并不会保存下来。镜像的上传过程与下载过程相反，客户端首先需要上传镜像各层文件，当层文件上传成功后，还需要更新 manifest 的签名信息，详细过程不再赘述。

在容器传输过程中，可能存在的安全风险包括镜像传输安全性问题（仓库与客户端之间是否通过安全加密传输协议进行传输）和镜像传输完整性问题（镜像传输过程可能存在中间人攻击篡改镜像或者下载文件不完整）。

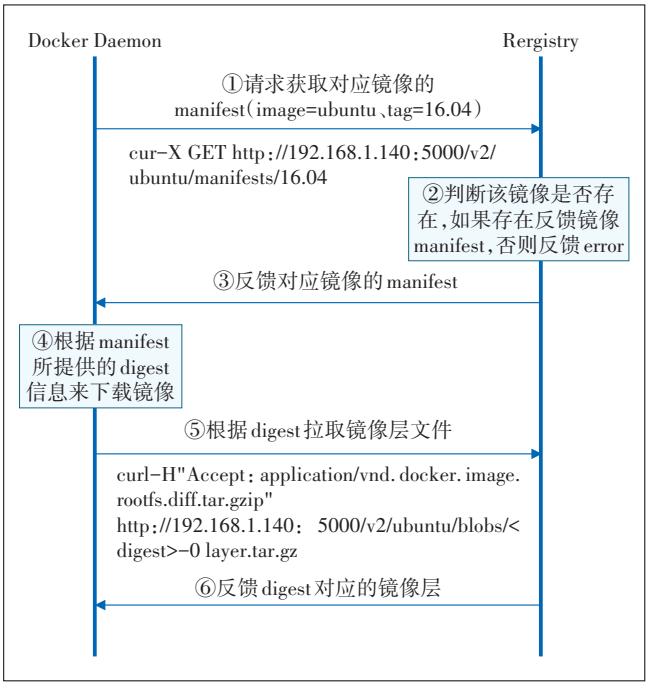

图3　Docker Daemon 从 Registry 下载镜像的过程

3.3 镜像在本地存储与风险

以 overlay2 存储驱动为例，容器镜像在本地存储也是将镜像元数据与镜像文件完全隔离开，这个理念

与镜像在仓库中存储的设计是一致的。镜像元数据存储路径是/var/lib/docker/image/overlay2,镜像文件存储路径是/var/lib/docker/overlay2。

镜像文件在本地的元数据包括repository、image、layer和distribution。由于镜像是以层的形式来存储的,所以repository与image这2类元数据没有物理上的镜像文件与之对应,而layer和distribution这2类元数据是有物理上的镜像层文件与之对应的。镜像在本地的实际存储位置是通过CacheID索引得到的,如图4所示。

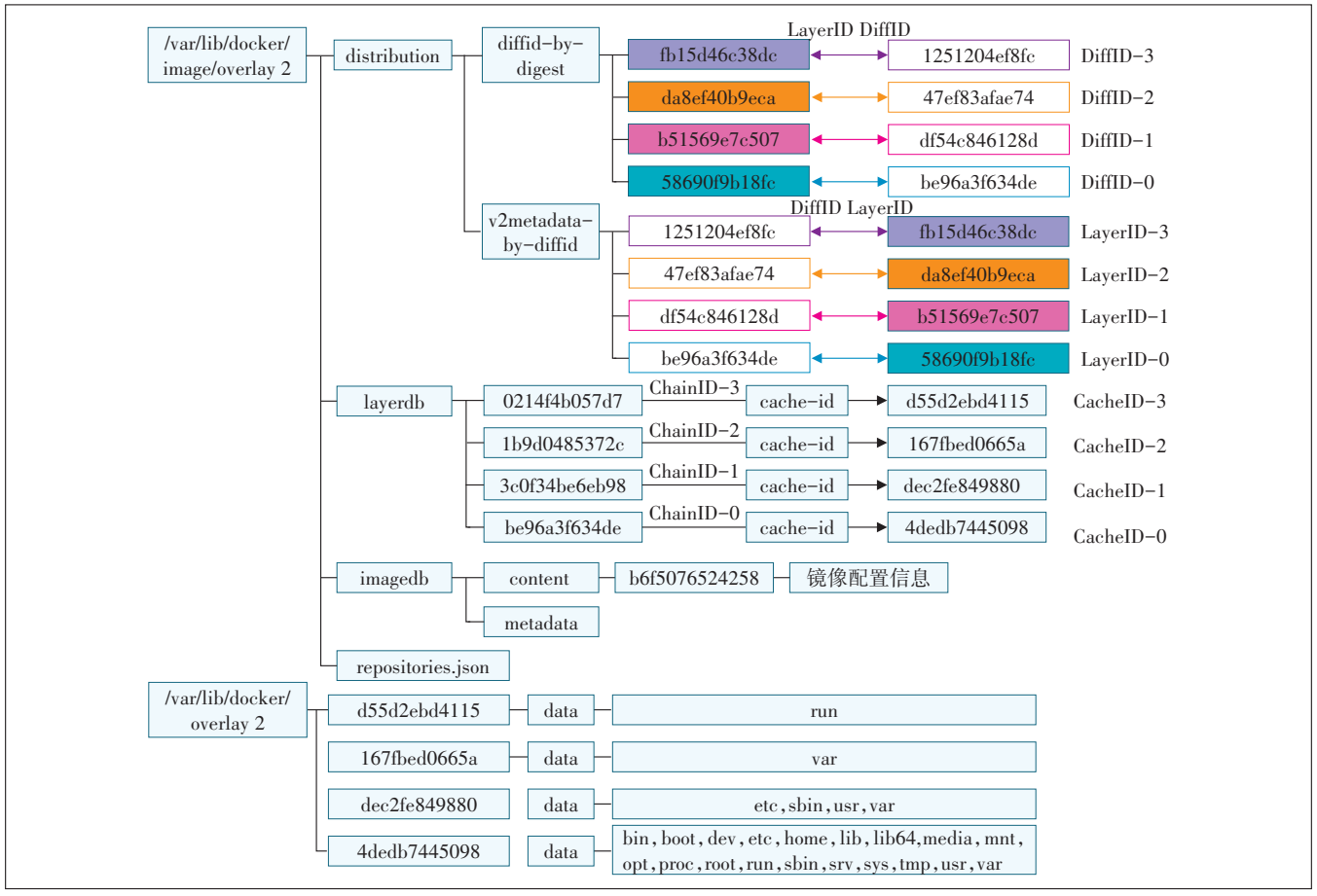

图4　镜像文件在本地存储

repository元数据中存放的是镜像仓库相关的信息,包括repository的名字、镜像名称、镜像版本和镜像ID(ImageID)等信息,详见repositories.json。

imagedb目录下保存镜像的元数据配置文件,b6f507652425文件中保存的内容与存储在镜像仓库中blobs下的相同文件名下保存的内容是一致的。

distribution目录下保存着LayerID和DiffID之间映射关系,LayerID为镜像层压缩数据的SHA256,即镜像在仓库中所存储的DiffID为下载到本地压缩状态下的镜像层的SHA256,可以通过以下步骤来验证DiffID的计算过程。

a)将镜像下载到本地。

```
# docker save ubuntu:16.04 > ubuntu.tar
```

b)解压缩ubuntu.tar。

```
# tar -xvf ubuntu.tar
```

c)计算不同镜像层的tar格式文件的SHA256数值,结果即为DiffID。

```
# sha256sum layer.tar
```

layerdb目录存储镜像的只读层和读写层文件,其标识为ChainID,其值根据当前层和所有祖先层的DiffID算得,计算公式如下:

$$\mathrm{ChainID}_n = \begin{cases} \mathrm{DiffID}_n & (n=0) \\ \mathrm{SHA256}\left(\mathrm{ChainID}_{n-1} + "" + \mathrm{DiffID}_n\right) & (n \geqslant 1) \end{cases} \quad (1)$$

目录layerdb/sha256/<ChainID>中存储了该层镜像对应的CacheID,CacheID是在宿主机上随机生成的一个UUID,与宿主机上的镜像层一一对应,在宿主机上的索引地址为/var/lib/docker/overlay2/<CacheID>。通过与仓库中的镜像信息对比,所有的镜像层文件被下载并且解压缩到本地目录。除此之外,还可以验证同一镜像在不同节点上的LayerID、DiffID、ChainID是完全一致的。

容器在本地存储过程中,除了存在安全漏洞、镜像木马病毒、信息泄露等风险外,还可能存在未经授权非法篡改的问题,启动容器时,系统不会对解压缩后的镜像文件进行二次检测,此时镜像文件可能被非法篡改。

4 镜像非法篡改与镜像漏洞扫描

4.1 一种非法篡改本地镜像的攻击过程

本文使用Dockerscan工具来模拟非法篡改本地镜像攻击,攻击者将木马文件植入本地镜像,当被篡改的镜像运行时,攻击者就会接收到反弹出的shell,从而达到控制服务器的目的。详细过程如下。

a)首先导出镜像文件。

docker save ubuntu:16.04 -o ubuntu-bak

b)向导出的镜像文件,添加反弹shell。

dockerscan image modify trojanize -l 192.168.1.142 -p 8888 ubuntu-bak -o ubuntu-attack

c)将原来的镜像文件替换为添加了反弹shell的镜像。

docker load -i ubuntu-attack.tar

d)在被监控端运行植入反弹shell的镜像,在监听端(192.168.1.142)进行监听。

完成上述操作后就可以在监控端(192.168.1.142)成功监控到受控主机的反弹信息。通过对比本地存储镜像的层文件,可以看出植入反弹shell的镜像,在最上层的镜像文件中多了一个usr目录,并且在该目录中存储名为reverse_shell.so的木马文件。此外,镜像环境变量发生的变化包括:LD_PRELOAD=/usr/share/lib/reverse_shell. so、REMOTE_ADDR=192.168.1.142、REMOTE_PORT=8888。由此可以判断,该镜像被非法植入了木马病毒。

4.2 容器镜像漏洞扫描

图5展示了Clair(一种Docker镜像漏洞扫描工具)的核心部分——ClairCore的功能架构。

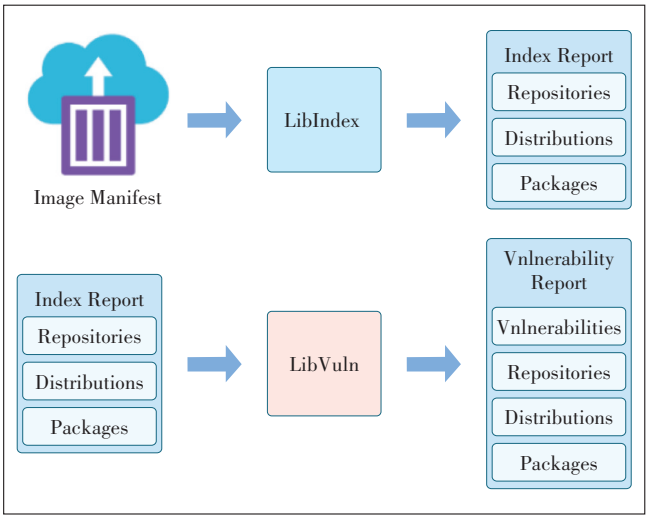

图5　ClairCore的功能架构图

ClairCore有2个核心组件LibIndex和LibVuln,当镜像的manifest传递到LibIndex时,LibIndex会根据manifest将编制镜像组成内容的索引,并创建索引报告。当索引报告传递到LibVuln时,LibVuln会检索镜像所存在的漏洞并生成脆弱性报告。为了使读者更好地理解LibIndex和LibVuln的工作原理,图6描述了LibInder中核心的索引器(Indexer)的工作流程,图7描述了LibVuln中的匹配器(Matcher)的数据流程图。

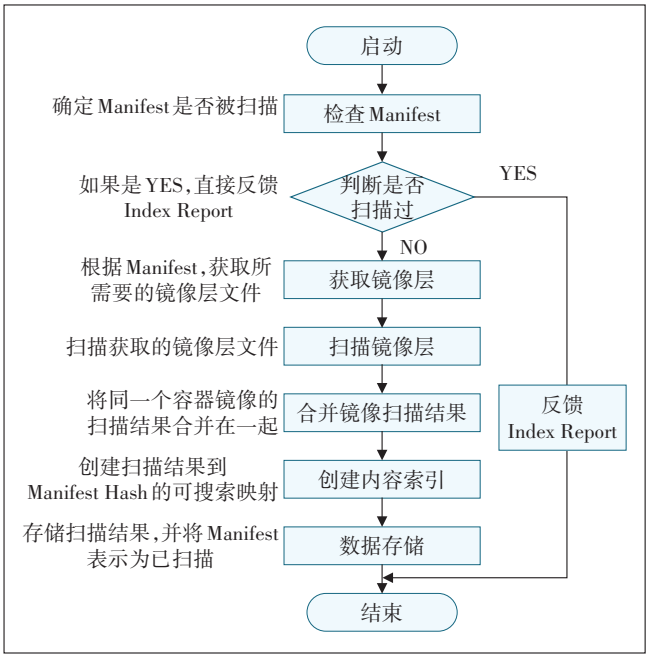

图6　Indexer工作流程图

在图7中,当IndexReport作为参数调用LibVuln的Scan方法时,将开启容器镜像漏洞的匹配过程。图7

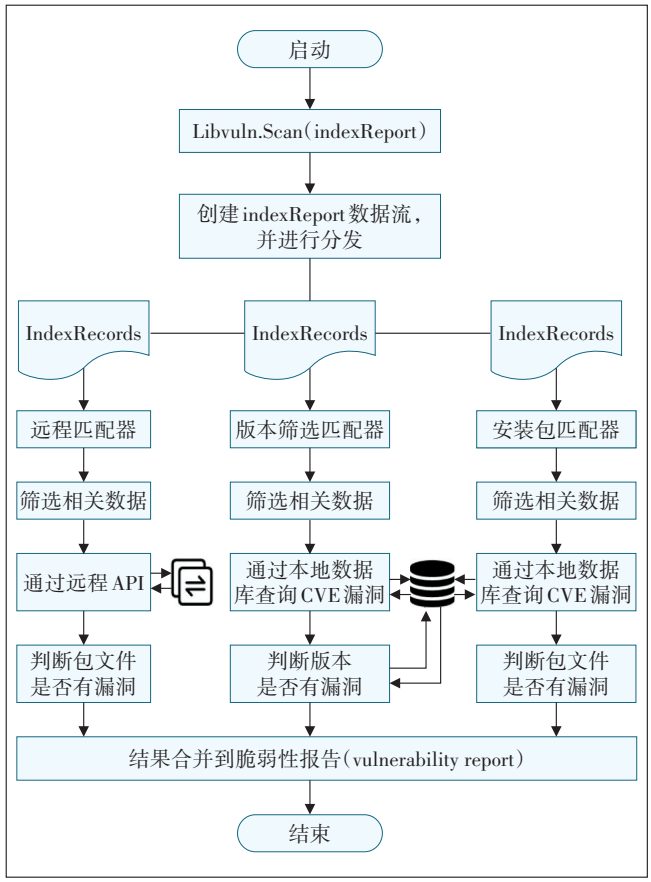

图7　Matcher数据流程图

展示的是一个可能出现的数据流程图，根据提供的IndexReport，数据将被解包到不同的流程中，匹配器将评估流中的每个数据，并确定是否存在漏洞，最终将匹配结果合并到脆弱性报告中，并将其反馈给客户端。

5 结束语

为了使读者更好地理解容器镜像安全扫描的过程，本文详细介绍了容器镜像在本地节点及镜像仓库的存储原理。无论是在本地存储还是在仓库中存储，镜像元数据与镜像文件是完全隔离存储的，这是镜像存储设计的一个基本思想。在不同镜像仓库中，镜像manifest存储路径和内容、blobs目录下的存储路径和内容完全一致。相同镜像在不同主机上的LayerID、DiffID、ChainID也是完全一致的，通过元数据信息获得CacheID，进而通过CacheID索引到镜像在本地的存储路径。

为了保证容器镜像的安全，容器镜像在存储过程中应具备保证数据完整性、机密性、可用性的能力，所以容器仓库应具备审核和加密存储镜像的能力。容器镜像扫描是发现容器安全漏洞、恶意代码的重要手段，但是镜像扫描不能解决容器生命周期中存在的所有问题，例如容器基础环境、容器编排器配置、共享资源、未知漏洞等安全问题。所以，只有将镜像安全扫描与容器运行时监控、容器安全编排等工具结合起来，才能实现容器全生命周期的安全管控。

参考文献：

[1] WIST K, HELSEM M, GLIGOROSKI D. Vulnerability analysis of 2500 docker hub images [M]//DAIMI K, ARABNIA H R, DELIGIANNIDIS L, et al. Advances in security, networks, and Internet of things. Cham: Springer, 2021: 307-327.
[2] 孙宏亮. Docker源码分析[M]. 北京：机械工业出版社，2015.
[3] 浙江大学SEL实验室. Docker容器与容器云[M]. 北京：人民邮电出版社，2015.
[4] 赵立农，曹莉，曾艺骁. Docker镜像安全深度扫描[J]. 通信技术，2021，54(2)：457-463.
[5] 王鹃，胡威，张雨菡，等. 基于Docker的可信容器[J]. 武汉大学学报(理学版)，2017，63(2)：102-108.
[6] Docker. Docker registry[EB/OL]. [2022-05-28]. https://docs.docker.com/registry/.
[7] Docker. Docker registry HTTP API V2 [EB/OL]. [2022-05-28]. https://docs.docker.com/registry/spec/api/.
[8] Github. Dockerscan [EB/OL]. [2022-05-28]. https://github.com/cr0hn/dockerscan.
[9] Docker. How the overlay driver works [EB/OL]. [2022-05-28]. https://docs.docker.com/storage/storagedriver/overlayfs-driver/.
[10] Github. Clair [EB/OL]. [2022-05-28]. https://github.com/quay/clair.
[11] Github. What is Clair[EB/OL]. [2022-05-28]. https://quay.github.io/clair/print.html.
[12] Github. What is ClairCore [EB/OL]. [2022-05-28]. https://quay.github.io/claircore/print.html.
[13] 刘晓毅，王进，冯中华，等. 容器云安全风险分析及防护体系设计[J]. 通信技术，2020，53(12)：3065-3071.
[14] 赵冠臣，王冬妮，刘至洋，等. 浅谈Docker容器技术[J]. 有线电视技术，2019，26(9)：85-88.
[15] 谢兆贤，倪冰雪，王若冰. 基于异常检测Docker容器的监控系统研究[J]. 计算机技术与发展，2022，32(6)：131-137.

作者简介：
丁攀，工程师，硕士，主要从事网络与信息安全研究工作；徐雷，高级工程师，博士，主要从事云计算、未来网络、网络与信息安全等新技术研究工作；刘安，工程师，硕士，主要从事网络与信息安全研究工作；苏俐竹，工程师，硕士，主要从事网络与信息安全研究工作。