计算机系统应用 ISSN 1003-3254, CODEN CSAOBN
Computer Systems & Applications,2022,31(12):301−308 [doi: 10.15888/j.cnki.csa.008814]

E-mail: csa@iscas.ac.cn
http://www.c-s-a.org.cn
Tel: +86-10-62661041

Docker 可信镜像源检测模型①

栗晓晗, 张新有

(西南交通大学 计算机与人工智能学院, 成都 611756)
通信作者: 张新有, E-mail: xyzhang@swjtu.edu.cn

摘　要: Docker 镜像是 Docker 容器运行的基础, 目前缺少完善的镜像安全检测方法, 导致容器运行时易受到容器逃逸、拒绝服务攻击等各种安全威胁. 为避免有毒镜像使用, 本文提出一种 Docker 可信镜像源检测模型 DTDIS (detect trusted Docker image source), 该模型使用可信密码模块 vTCM (virtual trusted cryptography module) 构建镜像基准值数据库, 检测本地镜像文件是否被篡改; 使用父镜像漏洞数据库扩展 Clair 镜像扫描器避免重复扫描; 结合文件度量信息、漏洞扫描信息判别 Docker 镜像源是否可信. 经云环境下实验证明, 该模型能够有效对 Docker 镜像进行安全评估, 保证用户使用可信镜像.

关键词: 漏洞扫描; Docker; Clair; 镜像依赖; vTCM; 网络安全; 镜像安全

引用格式: 栗晓晗,张新有.Docker 可信镜像源检测模型.计算机系统应用,2022,31(12):301–308. http://www.c-s-a.org.cn/1003-3254/8814.html

Detection Model for Docker Trusted Image Source

LI Xiao-Han, ZHANG Xin-You

(School of Computing and Artificial Intelligence, Southwest Jiaotong University, Chengdu 611756, China)

Abstract: Docker image is the operating basis of Docker containers. As robust methods of image security detection remain to be developed, containers are subject to various security threats, such as container escape and denial of service attacks, during their operation. To avoid the use of toxic images, this study proposes a detection model for trusted Docker image sources, namely detect trusted Docker image source (DTDIS). In this model, the virtual trusted cryptography module (vTCM) is used to build an image benchmark database and thereby detect whether the local image file has been tampered with. The parent image vulnerability database is utilized to extend the Clair image scanner and thus avoid repeated scanning. File measurement information and vulnerability scanning information are availed to determine whether the Docker image source is credible. Experiments in a cloud environment prove that the proposed model can effectively evaluate the security of Docker images and ensure that users use trusted images.

Key words: vulnerability scanning; Docker; Clair; image dependency; virtual trusted cryptography module (vTCM); network security; image security

1　引言

近年来 Docker 技术在开发者社区迅速升温, Docker 具有高移动性和可扩展性, 以简便的方式确保运行环境一致. 虚拟化技术的开发和使用呈爆炸式增长, 潜在的镜像安全问题也接踵而至. 如用户可以在 Docker hub 或其他镜像库中随意上传和下载任何镜像文件[1], 但据绿盟 2018 年的研究显示目前 Docker hub 上的 76% 镜像都存在安全风险. 镜像是容器运行的基础, 其安全极大程度影响容器安全, 镜像安全与用户的财产安全紧密相关, 而针对 Docker 镜像安全的研究还存在

① 基金项目: 国家自然科学基金 (61802319)
收稿时间: 2022-03-17; 修改时间: 2022-04-14; 采用时间: 2022-04-22; csa 在线出版时间: 2022-07-14

较大欠缺, 高效且安全的可信镜像源检测需求也逐步增加, 对镜像的安全检测工作迫在眉睫.

镜像文件包含高危漏洞、恶意代码或因本地镜像仓库缺乏安全防护导致镜像文件篡改等现象, 会导致容器逃逸、拒绝服务攻击[2] 等安全问题的产生, 因此镜像安全是保证可信容器的重要关注点. 本文提出一种可信镜像源检测模型, 它验证本地镜像是否被篡改并检测镜像漏洞, 输出镜像安全评估结果. 本文的主要贡献如下:

(1) 分析 Docker hub 云存储库中镜像分布情况, 建立涵盖官方镜像、验证镜像及社区镜像的父镜像数据集, 对其进行漏洞扫描构建父镜像漏洞数据库.

(2) 构建基准值数据库, 利用 vTCM 可信密码模块加密存储镜像基准值, 保证镜像从下载、本地存储、漏洞扫描至最终镜像运行过程中不被篡改.

(3) 扩展 Clair 镜像扫描器, 将父镜像漏洞数据库用于镜像漏洞扫描过程, 通过依赖关系避免重复扫描, 结合文件度量、漏洞扫描信息与自定义阈值对比从而判别可信镜像.

2 相关工作

本节概述 Docker 镜像安全工作研究现状及 Docker 镜像文件存储原理.

2.1 研究现状

Docker 自 2013 年面世至今有许多前辈为其安全问题作出贡献, 他们多将 Linux 容器防护技术运用至 Docker, 如 Amith 等[3] 将 Linux 安全模块 LSM 与 Docker 进行交互以提高 Docker 安全性; Jian 等[4] 提出一种基于命名空间状态的防御方法, 检测异常进程防止 Docker 容器逃逸; Loukidis-Andreou 等[5] 提出一种全自动化容器安全增强机制 Docker-sec, 利用创建容器时配置文件访问规则及容器运行时附加规则限制非授权容器进程通信, 保护容器免受伤害. 吉晨等[6] 将可信安全容器机制用于容器环境, 保护容器的数据安全问题. 上述对 Docker 安全的分析和研究解决了容器逃逸、拒绝服务攻击等问题, 在一定程度上提升了 Docker 使用安全性, 但都从容器角度防护忽略了 Docker 镜像的安全检测问题.

目前国内外对云环境下容器镜像安全研究较少, Jain 等[7] 介绍了现存镜像扫描器, 如 Anchroe 利用 Syft 模块生成软件安装包清单, Grppe 模块完成对容器镜像的扫描, 从而对容器镜像进行检查、分析和认证. Dcoker scan 扫描网络并尝试定位远程注册表信息、镜像中的敏感信息进行安全审查; Brady 等[8] 利用多阶段持续集成和持续部署技术来评估 Docker 镜像的安全性, 并在 Clair 基础上使用 AWS 环境搭建动态 API 扫描器实现镜像检测. Kwon 等[9] 提出镜像漏洞诊断系统 DIVDS, 利用 IVD 模块检测 Docker 镜像中潜存的漏洞, 使用 IVE 模块计算镜像漏洞分数, 通过检测分数决定是否允许使用该镜像建立可靠的 Docker 容器运行环境. 王娟等[10] 提取 Roofts 文件详细信息, 计算哈希值将其连接再次进行哈希运算使用 TPM 可信计算平台 bind 命令加密后保存. 在这种方式下若文件被篡改无法具体到某一个镜像, 而需销毁所有镜像, 且使用 TPM 芯片作为可信根对硬件环境要求较高.

综上, 现存镜像扫描框架均未考虑本地镜像文件的篡改风险, 且每次扫描镜像时从高到低依次扫描镜像层, 存在重复扫描导致扫描速度缓慢. 本文提出 DTDIS 模型预先构建基准值数据库, 通过度量保证所用镜像未被篡改; 结合父子依赖构建父镜像漏洞数据库, 利用父镜像漏洞数据库扩展 Clair 避免重复扫描根据扫描结果, 最终判断镜像源是否可信.

2.2 Docker 镜像

Docker 镜像有两种创建方式, 一是使用 Docker commit 将正在运行的容器中所有已安装应用和配置文件等打包为新的镜像, 二是使用 Dockerfile 将所需数据库、资源、程序、配置文件、配置参数等打包装入镜像当中. Docker 镜像生成时可利用已有镜像 (父镜像) 生成新镜像, 其中父镜像可以是 Docker 官方或其他官方所制作的基础镜像, 或为在基础镜像基础上制作的普通镜像.

镜像创建成功后可上传到 Docker 公司运行维护的镜像云存储库 Docker hub 中, 当需要某个本地不存在镜像时通过 Docker 引擎从云存储库中拉取, 下载到本地后存储镜像 Roofts 等文件, 镜像启动时在本地文件上添加读写层则变为容器 container. Docker 镜像生命周期如图 1 所示.

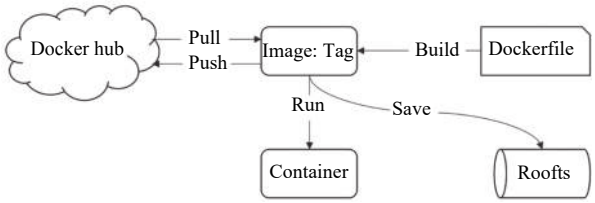

图 1 Docker 镜像生命周期

镜像文件是一个特殊的文件系统, 其中包含代码、系统环境、数据库和配置文件等软件运行所需的所有内容. 镜像由 4 部分组成: *manifest* 文件、镜像层文件、镜像配置文件、镜像索引序列号 (可选). 当需要某个镜像时使用 Docker pull 命令进行拉取, 拉取具体步骤如下:

1) 下载 *manifest* 文件:

$$manifest = manSvc.Get(tag) \quad (1)$$

2) 读取 *digest* 摘要:

$$digest = schema2Manifest(ref, mfst) \quad (2)$$

3) 下载镜像配置文件:

$$configJson=schema2ManifestImageConfig(imageDigest) \quad (3)$$

4) 使用加密存储镜像文件:

$$imageID = SHA256(digest.FormBytes(data)) \quad (4)$$

用户输入镜像名称和标签 (默认 latest) 至云存储库, 在 manifests 列表中解析获得对应 *manifest*, 读取 *digest* 数字摘要, *digest* 是镜像 *manifest* 经 SHA256 加密得到的摘要. 若某镜像层已存在直接指向本地文件, 不存在则向云存储库发送请求下载镜像配置文件和层文件, 镜像层下载完成后保存在本地仓库.

Docker 镜像按照配置文件中指定顺序分层堆叠, 利用联合文件技术将不同的镜像层整合为一个文件系统[11]. 镜像层可共享, 为防止文件被修改, 引入写时复制机制 (copy on write, COW) 构建镜像. Docker 镜像文件由“层”堆砌而成, 下层存放多个只读层文件, 当容器启动时添加一个轻量可重用读写层在顶层[12], 容器对文件所做的任何操作都存储在该读写层中, 镜像运行时在静态文件上添加一个可读写的容器层, Python 镜像分层示意如图 2 所示.

3 DTDIS 模型

本节介绍 DTDIS 模型架构及具体工作流程.

3.1 模型架构

DTDIS 模型由文件度量模块和漏洞扫描模块组成, 用户使用 Docker pull 命令从 Docker hub 中拉取 Docker 镜像, Docker 引擎下载完成后存储到本地. 文件度量模块负责首次加载镜像时, 计算基准值并将哈希值等相关信息经 vTCM 加密存储后存入基准值数据库, 容器运行前再次进行文件度量并输出度量信息. 漏洞扫描模块利用父镜像漏洞数据库避免重复扫描, 对比所设置 CVE 数据库漏洞数据输出镜像漏洞信息, 最终结合漏洞信息、度量信息对镜像计算镜像可信评分判定镜像是否可信, DTDIS 模型架构图如图 3 所示.

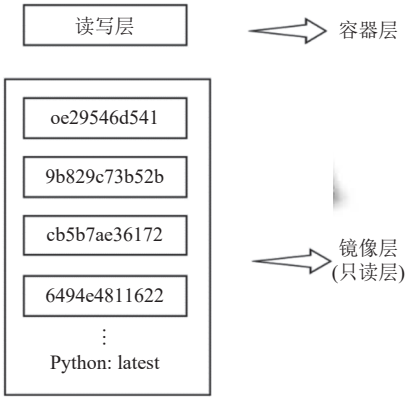

图 2 Docker 镜像分层示意图

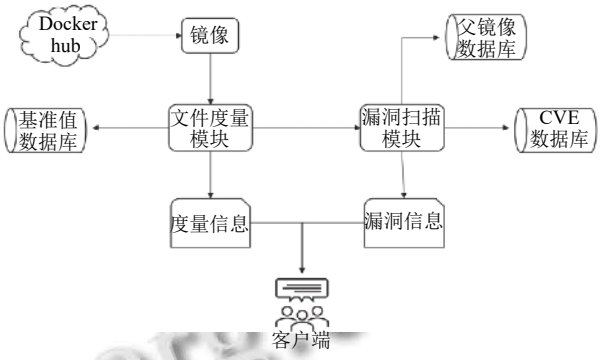

图 3 DTDIS 模型架构图

3.2 工作流程

图 4 展示 DTDIS 工作流程, 具体步骤可概括为以下:

1) 用户使用 Docker pull [image name] 命令从 Docker hub 云仓库请求下载父镜像数据集名单中 Docker 镜像.

2) Docker 引擎解析 Docker pull 命令, Docker hub 匹配成功后允许下载执行, 镜像下载完成保存在本地镜像仓库.

3) 对于步骤 2) 下载的镜像使用 Clair 进行镜像扫描, 将扫描结果存入父镜像漏洞数据库中.

4) 用户使用 Docker pull [image name]:[tag] 从 Docker hub 云仓库请求所需 Docker 镜像.

5) 重复步骤 2).

6) 将步骤 5) 获取的镜像加载至 DTDIS, 若该镜像为初次加载则文件度量模块提取 Roofts 文件, 使用算

法 1 对其进行加密存储.

7) 对于步骤 6) 加载的镜像, 提取 Roofts 文件元数据依次判断镜像层是否存在于父镜像漏洞数据库, 若存在则直接获取漏洞信息.

8) 若步骤 7) 发现镜像层在父镜像漏洞数据库未找到, 则使用算法 2 进行漏洞扫描, 与漏洞数据库中易受攻击软件包依次匹配, 输出漏洞信息.

9) 运行镜像前使用文件度量模块对镜像进行度量, 得到度量信息.

10) 使用式 (1) 对步骤 9) 所得度量信息与步骤 7) 或 8) 所得漏洞信息进行计算输出镜像可信分数 $image_{score}$.

11) 将阈值与步骤 10) 所得 $image_{score}$ 进行比较后, 将可信镜像提供给用户.

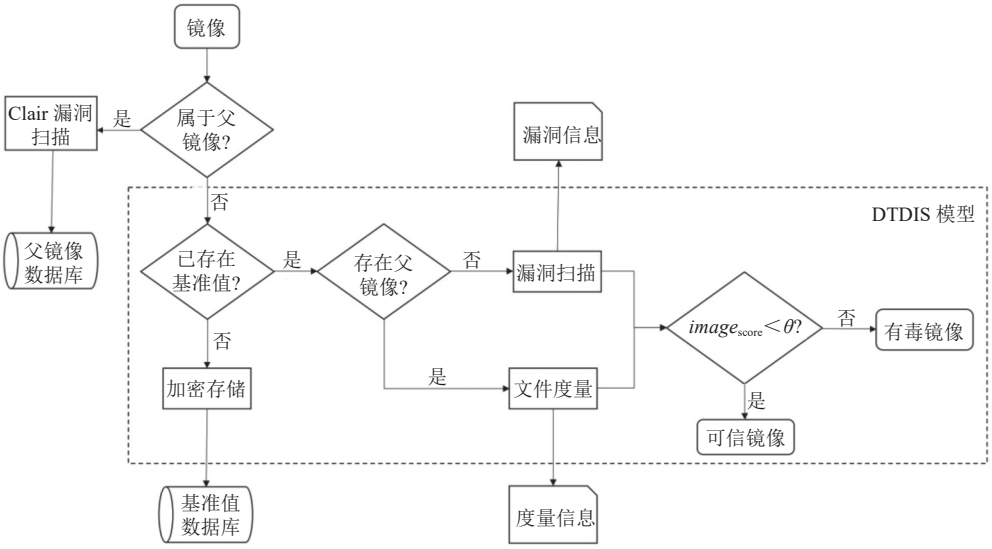

图 4　DTDIS 工作流程图

4　DTDIS 核心模块

本节介绍 DTDIS 模型核心, 文件度量模块和漏洞扫描模块实现细节.

4.1　文件度量模块

Docker 使用 3 个文件存储镜像, repositories.json 存储仓库名称、镜像版本号和镜像 ID, imagedb 存储镜像的元数据配置文件, layerdb 存储镜像分层信息.

镜像分层存储, 按照由低到高的顺序存储在 Roofts 文件中, Layer.DiffID 用来检测镜像层文件的完整性, 据式 (6) 为对镜像未压缩的元数据内容进行 SHA256 所得[13]. 如图 5 所示 Python 镜像层 Layer.DiffID 值为 119360…127bce 等.

```
"RootFS": {
    "Type": "layers",
    "Layers": [
        "sha256:11936051f93baf5a4fb090a8fa0999309b8173556f7826598e235e8a82127bce",
        "sha256:31892cc314cb1993ba1b8eb5f3002c4e9f099a9237af0d03d1893c6fcc559aab",
        "sha256:8bf42db0de72f74f4ef0c1d1743f5d54efc3491ee38f4af6d914a6032148b78e",
        "sha256:26a504e63be4c63395f216d70b1b8af52263a5289908df8e96a0e7c840813adc",
        "sha256:c3a0d593ed24e738aec3b1e61662f696311ae99941c45c78d281ebc3732bdcb0",
        "sha256:aedcb370b058ce0937aec788703b6ef40780f673de1238d38334fa367e83f21e",
        "sha256:7e7decd61f68a8a9c313d41d19fc38435cd0aeecdc5252d1709d0ab902a84e92",
        "sha256:00901a4c6fc7aa93bc6e6c448c75601bc0df03103ed4414a3de891193d4c697f",
        "sha256:db8d0fe6cf959ca43a2c865dec6b086dd55489b0a9ab1a484a787ec42d3c8875"
    ]
},
```

图 5　Python 镜像层示例图

Docker 内容寻址机制采用 ChainID 为索引 ID, 每个 ChainID 都对应一个目录, 每个目录代表一个镜像层, 且该目录中包含一个 parent 文件存储其父层的 ChainID. 如果当前层是最底层, 不存在父层镜像则使

用式 (7), 如果当前层存在父层镜像则使用式 (8) 计算.

$$ImageID = SHA256(ConfigJson) \tag{5}$$

$$Layer.DiffID = SHA256(Uncompressed\_layer\_data) \tag{6}$$

$$Layer.ChainID(Layer0) = Layer.DiffID(Layer0) \tag{7}$$

$$Layer.ChainID = SHA256hex(Layer.ChainID(layerN-1)) + \text{“.”} + DiffID(layerN)) \tag{8}$$

本文使用 Cube-vTCM 运行在 Linux 内核中实现可信环境搭建, 使用分层哈希值存储逐一对应镜像层, 具体文件度量模块原理如算法 1 展示.

输入当前镜像的 Roofts 文件, R 代表镜像文件的详细信息 R={R[0], R[1], R[2], ⋯, R[K]}, 其中 K 为镜像层数; R.layer_id 代表当前镜像层 ID; R.layer_chain_id 代表当前镜像层的父镜像层 id; R.image_pkg _name 表示镜像内应用软件包名称; R.image_version 代表镜像内应用软件包版本.

算法 1. 文件度量算法

输入: 镜像 Roofs 文件 R
输出: 镜像基准值 result

```
1)  for i in range(0, K) do
      //获取层 id
2)    P1 = hashlib.sha1(R[i].layer_id).hexdigest()
      //获取寻址索引 id
3)    P2 = hashlib.sha1(R[i].layer_chain_id).hexdigest()
      //获取软件包名称
4)    P3 = hashlib.sha1(R[i].image_pkg_name) .hexdigest()
      //获取软件包版本
5)    P4 = hashlib.sha1(R[i].image_version).hexdigest()
6)    str = P1 + P2 + P3 + P4
7)    ref_value = hashlib.sha1()
8)    result = ref_value.update(str).hexdigest()
9)    result → vTCM //存储基准值
10) endfor
```

文件度量模块存储基准值及镜像度量, 当镜像首次载入文件度量模块, 使用 SHA-1 算法对各镜像层进行散列运算, 连接后二次哈希运算所得值使用 vTCM 加密存储作为基准值存入数据库中, 漏洞扫描结束后进行文件度量.

4.2 漏洞扫描模块

基于父镜像构建新镜像虽减少文件存储空间、加快镜像构建速度, 但这种可重用性导致漏洞依赖传递, 若基础镜像中存在镜像漏洞会导致所有以它为基础的镜像也存在同样的安全问题. 本文利用这种依赖传递关系, 构建父镜像漏洞数据库避免重复扫描, 提取镜像文件元数据与 CVE 漏洞数据库信息进行对比, 算法 2 具体实现过程如下.

输入当前镜像的 Roofts 文件, 其表示方法同文件度量模块. DI 代表数据库中存储的镜像层信息, DI={DI[0], DI[1], DI[2], ⋯, DI[N]}, 其中 N 为镜像层数, 定义 DI.layer_id 代表镜像层 ID; DI.layer_cve_number 代表镜像内漏洞 CVE 唯一编号; DI[j].layer_cve_desc 代表 CVE 漏洞描述信息.

漏洞扫描模块利用 Clair 的 CVE 数据库模块从乌班图 CVE 追踪器, Debian 漏洞追踪器, 红帽安全数据集等公开 CVE 字典库定时收集漏洞, 更新漏洞数据库. 定义 V 代表 CVE 漏洞数据库中的存在漏洞软件包条目, V={V[0], V[1], V[2], ⋯, V[M]}, 其中 M 代表数据库中漏洞的总数. 定义 V.cve_number 表示漏洞的 CVE 唯一编号; V.cve_description 表示漏洞的详细描述; V.cve_pkg_name 当前漏洞所属软件包名称; V.cve_pkg_version 当前漏洞所属软件包版本.

算法 2. 漏洞扫描算法

输入: 镜像 Roofs 文件 R, 父镜像漏洞 DI, CVE 漏洞 V
输出: 镜像漏洞信息 vul_Info

```
1)  for i in range(0, K) do
2)    for j in ranger(0, N) do
        //判断是否存在父镜像信息
3)      if R[i].layer_chain_id == DI[j].layer_id
4)  vul_Info[index]=DI[j].layer_cve_number, DI[j].layer_cve_ desc
5)          index++;//扫描所有镜像层
6)        endif
7)    endfor
        //匹配软件包名称及版本
8)    for t in range(0, M) do
9)      if V[t].cve_pkg_name == R[i].image_pkg_name and
      V[t].cve_p kg_version == R[i].image_version
10)       vul_Info [index] = V[t].cve_number, V[t].cve_des cription
11)       index++;
12)     endif
13)   endfor
14)  endfor
15)  if index == 1 //判断是否存在漏洞信息
16)    vul_Info [index] = “The image has no vulnerabilities”
17)  endif
18)  Output vul_Info
19)  end
```

镜像经漏洞扫描后输出所包含的 CVE 漏洞详细信息, 结合文件度量信息根据式 (9) 计算对镜像进行可信评分, 其中r_{sv}为漏洞扫描模块所得镜像包含漏洞总数, w_vul_{sv}为表 1 中漏洞对应权重[11], w_dat_{sv}为表 2 中包含漏洞的软件包最近更新时间权重, v_{init}为文件度量模块数据库中存储的镜像初始基准值, v_{cur}为镜像当前基准值, w_{ref}为表 3 中文件度量权重.

表 1 CVE 漏洞权重w_vul_{sv}

CVE严重性	权重
未知	0.0
可忽略	0.5
较低	2.0
中等	5.45
严重	7.95
危急	9.5

表 2 最近更新时间权重w_dat_{sv}

更新时间	权重
一周内	0.0
两周内	0.5
一月内	1.0
半年内	1.5
一年内	2.0
一年以上	2.5

表 3 基准值权重w_{ref}

基准值	权重
与初始一致	0.0
与初始不同	9.5

$$image_{score}=\sum_{i=1}^{r_{sv}}(w\_vul_{sv}+w\_dat_{sv})+100\cdot(v_{init}-v_{cur})\cdot w_{ref} \tag{9}$$

通过式 (9) 计算镜像可信得分, 因不同应用场景对安全程度规定不同, 故本模型采用自定义阈值 θ, 将其与镜像可信得分比较, 根据式 (10) 比较阈值 θ 和 $image_{score}$, 若$image_{score}$高于 θ 则认为该镜像不可信, 根据式 (11) 比较阈值$image_{score}$和 θ, 若$image_{score}$低于或等于 θ 则认为该镜像可信, 阈值 θ 可由用户设置.

$$image_{score}=\sum_{i=1}^{r_{sv}}(w\_vul_{sv}+w\_dat_{sv})+100\cdot(v_{init}-v_{cur})\cdot w_{ref}>\theta \tag{10}$$

$$image_{score}=\sum_{i=1}^{r_{sv}}(w\_vul_{sv}+w\_dat_{sv})+100\cdot(v_{init}-v_{cur})\cdot w_{ref}\leqslant\theta \tag{11}$$

对于不同安全等级漏洞给予不同权重, 且保留白名单功能, 最终可信值计算时忽略白名单中所出现的 CVE 漏洞. CVE 漏洞权重取值参考文献 [11] 与 CVSS (安全漏洞评分系统) 分数设定, 文献 [9] 指出镜像更新时间与安全程度呈正相关故表 2 权重取值如下, 文献 [10] 指出本地镜像文件被篡改具有安全的严重隐患, 故本文将镜像文件篡认作最高等级危急事件, 设置基准值权重为表 3 所示.

5 实验结果与分析

本节介绍 DTDIS 模型的实验结果与分析, 包括实验环境, 镜像集分布及功能架构等相关细节实现.

5.1 实验环境

Docker 容器大多运行在云环境下, 本文使用阿里云 ECS 服务器 CentOS 7 系统, 利用 Cube-vTCM 搭建可信环境进行单机仿真实验, 具体实验环境如表 4 所示.

表 4 实验环境

名称	版本
ECS服务器	2 vCPU 2 GiB
系统版本	CentOS 7
Linux内核	3.10.0 x86_64
Docker版本	20.10.11
Cube-vTCM	1.3

5.2 镜像数据集构建

截止到 2021 年 12 月 Docker hub 拥有约 850 万个镜像, 分别存储在官方仓库、验证仓库和社区仓库, 具体分布如表 5 所示.

表 5 Docker hub 镜像仓库分布

仓库类型	镜像数量	占有率 (%)
Official	171	0.002 018
Verified	8 776	0.103 567
Community	8 464 811	99.894 415
总计	8 473 758	100

本文的数据集涵盖官方、社区和验证 3 种镜像, 且为保证高效, 同一个存储库中的镜像只拉取最新更改版本, 有些许镜像在下载过程中存在未知错误无法使用:

1) 如“mcr.microsoft.com/hpcacm”属于验证仓库镜像, 通过 Docker hub 可视化界面搜索可得, 但无法使用 Docker pull 命令完成下载任务, 抛出错误.

Pulling repository mcr.microsoft.com/hpcacm

Error: image hpcacm:latest not found

同样的错误出现在官方仓库共计 9 个镜像无法使用 Docker pull 命令下载.

2) 如“Docker/cheers”属于官方仓库镜像, 因为其最新版本号不是 latest 需要指定版本号下载, 使用 Docker pull 命令拉取会抛出以下错误.

Trying to pull repository docker.io/ docker/cheers

manifest for docker.io/cheers:1atest not found

同样的错误出现在验证仓库中共计 18 个镜像无法使用 Docker pull 命令下载, 忽略这些镜像最终镜像数据集构成如表 6.

参与 Verified Publisher 计划的用户所发布的镜像为验证镜像, 该仓库镜像未有详细分类. 最终父镜像集如表 6 所示, 使用 762 社区镜像、162 官方镜像及 8 620 个验证镜像共 9 544 个镜像构成, 依次下载镜像从 Roofs 文件中提取元数据扫描后构建父镜像漏洞数据库.

5.3 实验结果与分析

由于镜像可信为相对概念, 在不同使用场景用户对镜像安全度要求不同, 故该模型采用用户自定义阈值判断镜像是否可信. 采取与文献 [11] 相同的阈值, 设置 θ 为 200. 将 CVE-2014-4210、CVE-2019-19814 等 100 个高危漏洞放入镜像文件使其成为有毒镜像, 使用 DTDIS 模型进行检测结果如图 6 所示.

表 6 数据集详情表

镜像类型	Community	Official	Verified
Analytics	53	4	—
Application Frameworks	50	21	—
Application Infrastructure	56	14	—
Application Services	35	25	—
Base Images	177	16	—
Databases	36	13	—
DevOps Tools	40	9	—
Featured Images	2	5	—
Messaging Services	7	4	—
Monitoring	51	1	—
Operamming Systems	1	15	—
Programming Languages	21	21	—
Security	16	0	—
Storage	18	4	—
Other	10	35	8 620
总计	762	162	8 620

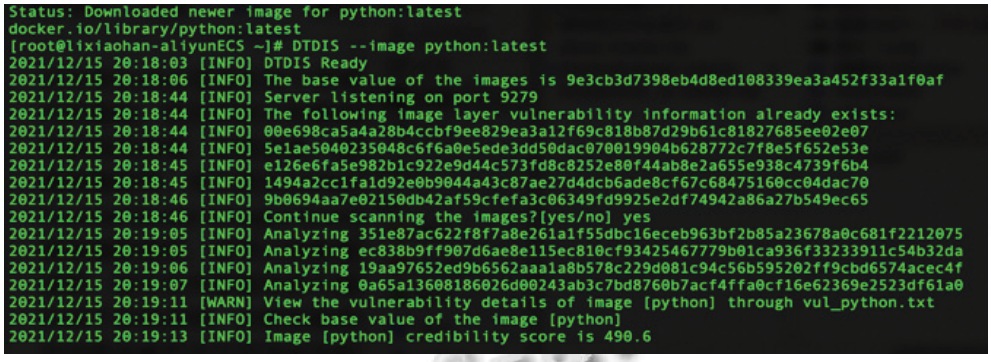

图 6 DTDIS 模型扫描结果

由图 6 可知 Python 镜像存储在本地后载入 DTDIS 模型, 利用 vTCM 加密存储后文件度量值为“9e3cb3d7…”, 因 Python 镜像文件与 Linux 镜像层部分重合, 且 Linux 镜像已存入父镜像漏洞数据库故“00e698ca…”, “5e1ae504…”等 5 个镜像层无需再次进行漏洞扫描, 表 7 展示部分镜像漏洞信息.

表 7 Python 镜像漏洞表

CVE名称	CVE严重性	安装包信息	CVE详细描述
CVE-2019-19814	Critical	Linux	In the Linux kernel 5.0.21, mounting a crafted f2fs filesystem image can cause __remove_dirty_ segment slab-out-of-bounds write access because an array is bounded by the number of dirty types (8) but the array index can exceed this https://security-tracker.debian.org/tracker/CVE-2019-19814
CVE-2021-33574	High	glibc	The mq_notify function in the GNU C Library (aka glibc) versions 2.32 and 2.33 has a use-after-free. … https://security-tracker.debian.org/tracker/CVE-2021-33574

由表 7 可知 Python:latest 镜像中包含 Linux、glibc 等软件包, 软件包中含有危急漏洞 CVE-2019-19814、高危漏洞 CVE-2021-33574 等, 通过 CVE 详细描述可获取漏洞详细信息. 漏洞扫描结束后将漏洞信息结合

文件度量信息, 由图 6 最后一行所示, 经计算 Python: latest 镜像可信得分为 490.6, 高于当前阈值 θ, 则判定其是一个有毒镜像.

6 总结

本文提出 DTDIS 模型判断 Docker 镜像源是否可信, 该模型主要包括文件度量模块和漏洞扫描模块. 文件度量模块构建镜像基准值数据库, 使用 vTCM 进行加密、度量保证所用镜像未被篡改. 漏洞扫描模块在 Clair 镜像扫描器基础上开发, 构建父镜像漏洞数据库, 通过父子依赖避免 CVE 漏洞重复扫描, 并通过自定义阈值判断镜像源是否可信. 在云服务器验证 DTDIS 模型效果, 结果表明 DTDIS 模型可以在较短时间内分析镜像, 判断 Docker 镜像源是否可信, 确保用户使用可信镜像从而降低容器逃逸、信息泄露等安全危害发生. 本文只考虑镜像静态安全, 未来将结合动态安全分析技术进一步保证 Docker 容器的安全使用.

参考文献

1 Combe T, Martin A, Di Pietro R. To Docker or not to docker: A security perspective. IEEE Cloud Computing, 2016, 3(5): 54–62. [doi: 10.1109/MCC.2016.100]

2 List M. Using Docker compose for the simple deployment of an integrated drug target screening platform. Journal of Integrative Bioinformatics, 2017, 14(2): 20170016.

3 Amith Raj MP, Kumar A, Pai SJ, *et al*. Enhancing security of Docker using Linux hardening techniques. 2016 2nd International Conference on Applied and Theoretical Computing and Communication Technology (iCATccT). Bangalore: IEEE, 2016. 94–99.

4 Jian ZQ, Chen L. A defense method against Docker escape attack. Proceedings of the 2017 International Conference on Cryptography, Security and Privacy. Wuhan: ACM, 2017. 142–146.

5 Loukidis-Andreou F, Giannakopoulos I, Doka K, *et al*. Docker-sec: A fully automated container security enhancement mechanism. 2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS). Vienna: IEEE, 2018. 1561–1564.

6 吉晨, 石勇, 戴明, 等. 基于轻量级虚拟化环境的可信多级安全容器机制. 计算机应用研究, 2017, 34(6): 1770–1773. [doi: 10.3969/j.issn.1001-3695.2017.06.037]

7 Jain V, Singh B, Khenwar M, *et al*. Static vulnerability analysis of Docker images. IOP Conference Series: Materials Science and Engineering. 2021, 1131: 012018. [doi: 10.1088/1757-899X/1131/1/012018]

8 Brady K, Moon S, Nguyen T, *et al*. Docker container security in cloud computing. 2020 10th Annual Computing and Communication Workshop and Conference (CCWC). Las Vegas: IEEE, 2020. 975–980.

9 Kwon S, Lee JH. DIVDS: Docker image vulnerability diagnostic system. IEEE Access, 2020, 8(10): 42666–42673.

10 王鹃, 胡威, 张雨菡, 等. 基于 Docker 的可信容器. 武汉大学学报 (理学版), 2017, 63(2): 102–108.

11 Kwon S, Lee JH. DIVDS: Docker image vulnerability diagnostic system. IEEE Access, 2020, 8: 42666–42673. [doi: 10.1109/ACCESS.2020.2976874]

12 李毅伦, 宋虹, 王大成, 等. 基于应用程序分层技术的镜像管理机制研究. 计算机应用与软件, 2018, 35(2): 22–29. [doi: 10.3969/j.issn.1000-386x.2018.02.004]

13 Zheng Y, Dong WY, Zhao JT. ZeroDVS: Trace-ability and security detection of container image based on inheritance graph. 2021 IEEE 5th International Conference on Cryptography, Security and Privacy (CSP). Zhuhai: IEEE, 2021. 186–192.

(校对责编: 孙君艳)