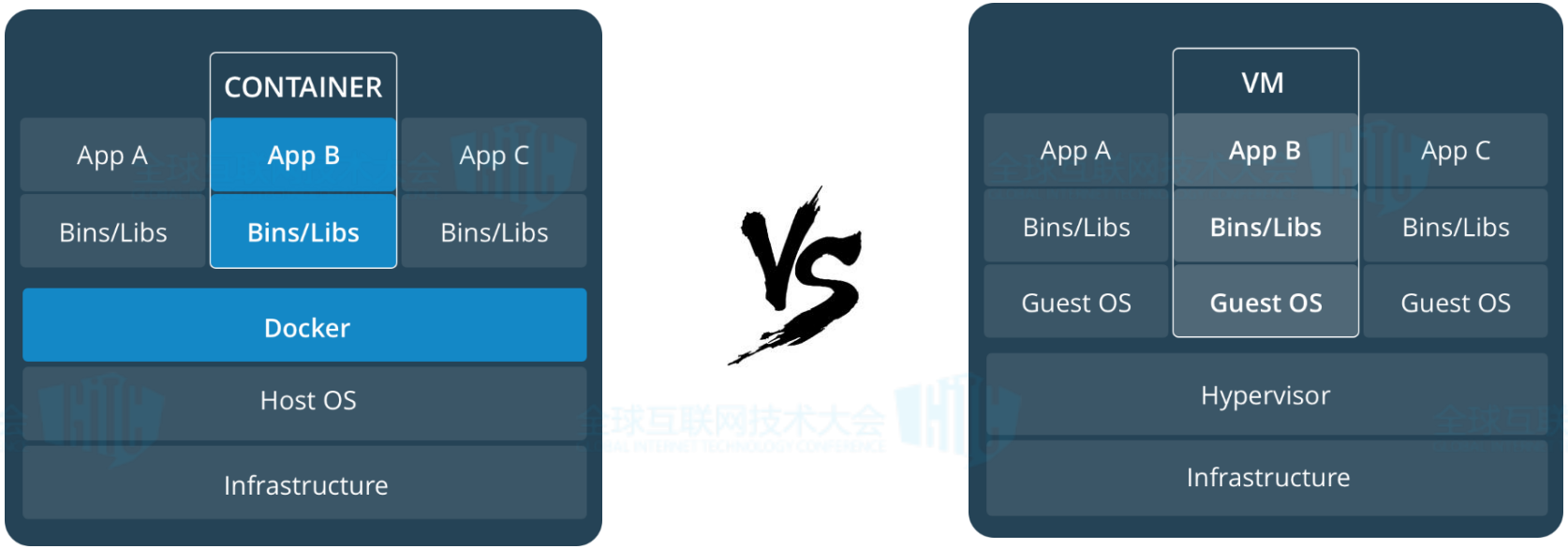
Docker安全实践探索

袁曙光 联众游戏 安全运维总监

Docker面临的安全挑战

Docker VS Vm

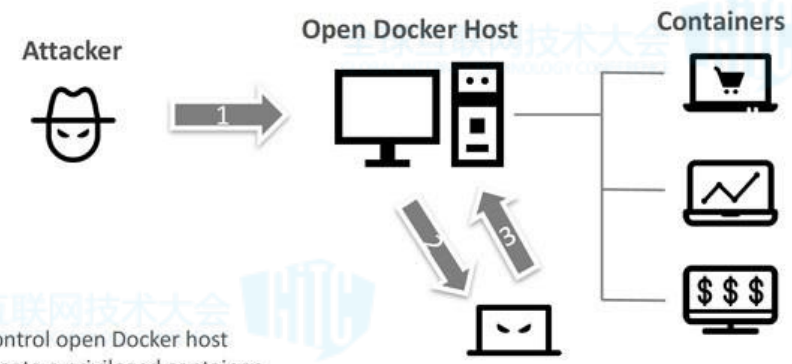


容器是应用层的一个抽象，它将代码和依赖项集成在一起。多个容器可以在同一台机器上运行，并与其他容器共享操作系统内核，每个容器都以用户空间中的隔离进程运行。容器占用的空间少于虚拟机（容器映像的大小通常为几十MB），几乎立即启动。

虚拟机（VM）是将一个服务器转换为许多服务器的物理硬件的抽象。虚拟机管理程序允许多个虚拟机在单个计算机上运行。每个虚拟机包括操作系统的完整副本，一个或多个应用程序，必要的二进制文件和库 - 占用数十GB。VM也可能启动缓慢。

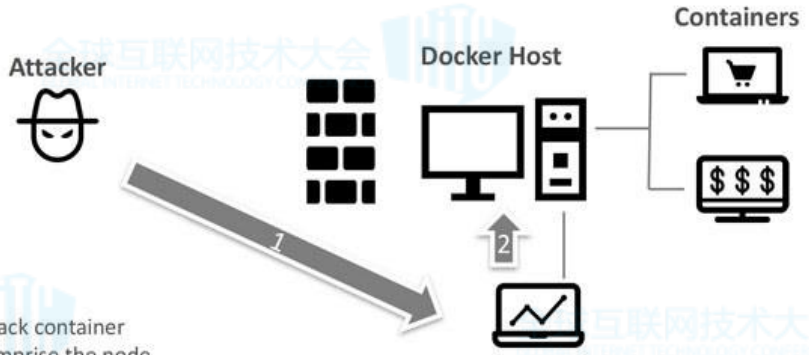
Docker面临主要攻击方式

Attack – Opened Node from Outside



1. Control open Docker host
2. Create a privileged container
3. Compromise the host

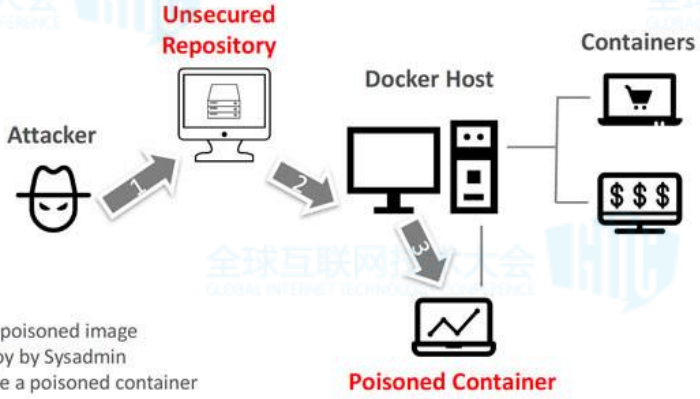
Attack – Opened Node from Inside



1. Attack container
2. Comprise the node

Because container can access management interface from container network. ☺

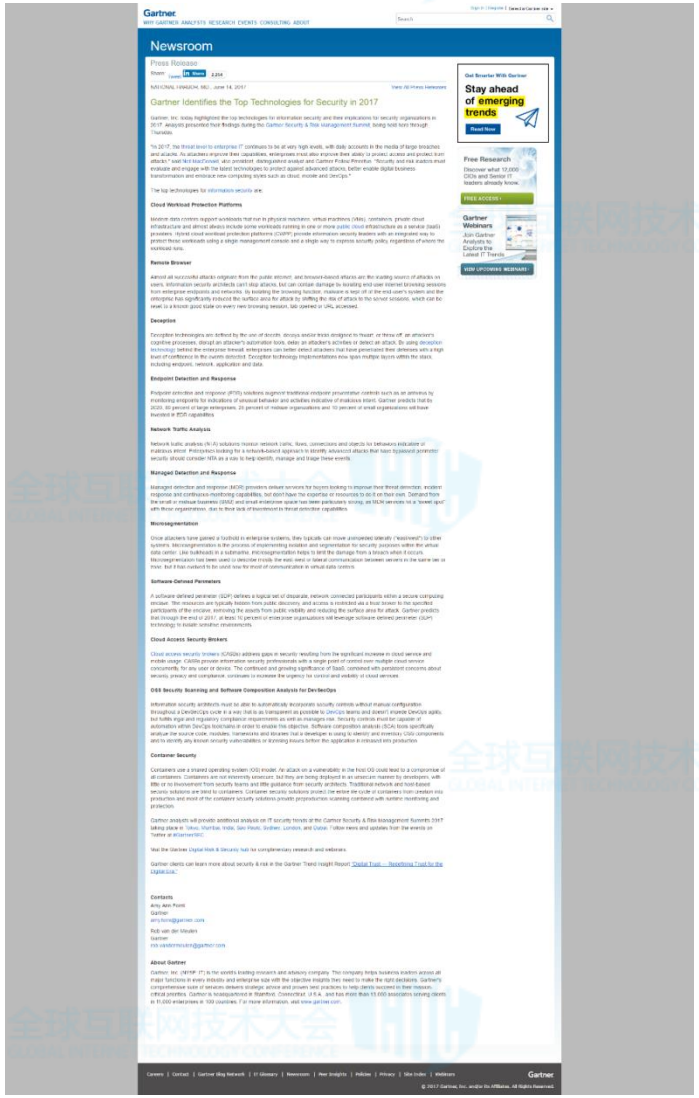
Attack – Poisoned image



1. Push poisoned image
2. Deploy by Sysadmin
3. Create a poisoned container

图片来源：趋势科技

Gartner研究结果



Gartner在2017年确定了最高安全技术
Gartner公司今天突出介绍了信息安全的顶级技术及其对安全组织的影响。分析师在**Gartner安全与风险管理峰会**期间介绍了的研究结果。

容器安全

容器使用共享操作系统（OS）模型。对主机操作系统中的漏洞的攻击可能导致所有容器的妥协。***容器本身并不安全，但由开发人员以不安全的方式部署，安全团队很少或根本没有参与，安全架构师也没有指导***。传统的网络和基于主机的安全解决方案对容器是无视的。集装箱安全解决方案保护集装箱的整个生命周期不被创造到生产，而大多数集装箱安全解决方案提供了预生产扫描和运行时监控和保护。

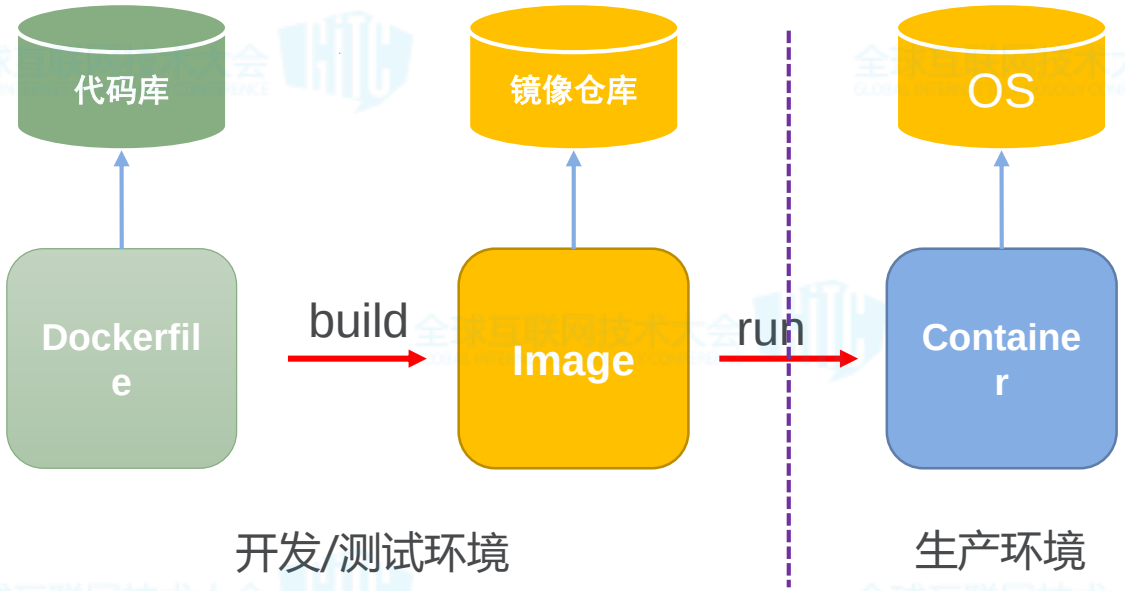
http://www.gartner.com/newsroom/id/3744917

Docker的生命周期

- Dockerfile:用于创建image镜像的模板文件
- Image:镜像文件，对比PC端的概念，我们可以把它理解为服务器端的可执行软件包。一旦打包生成，如存在安全问题，那这些问题也被一并打包，最后导致安全事件
- Container：运行起来的image文件就是容器了，从外来看就是一个应用，可对外提供服务了

Docker安全防护本质上是保证docker镜像在创建、存储、传输、运行过程中的安全

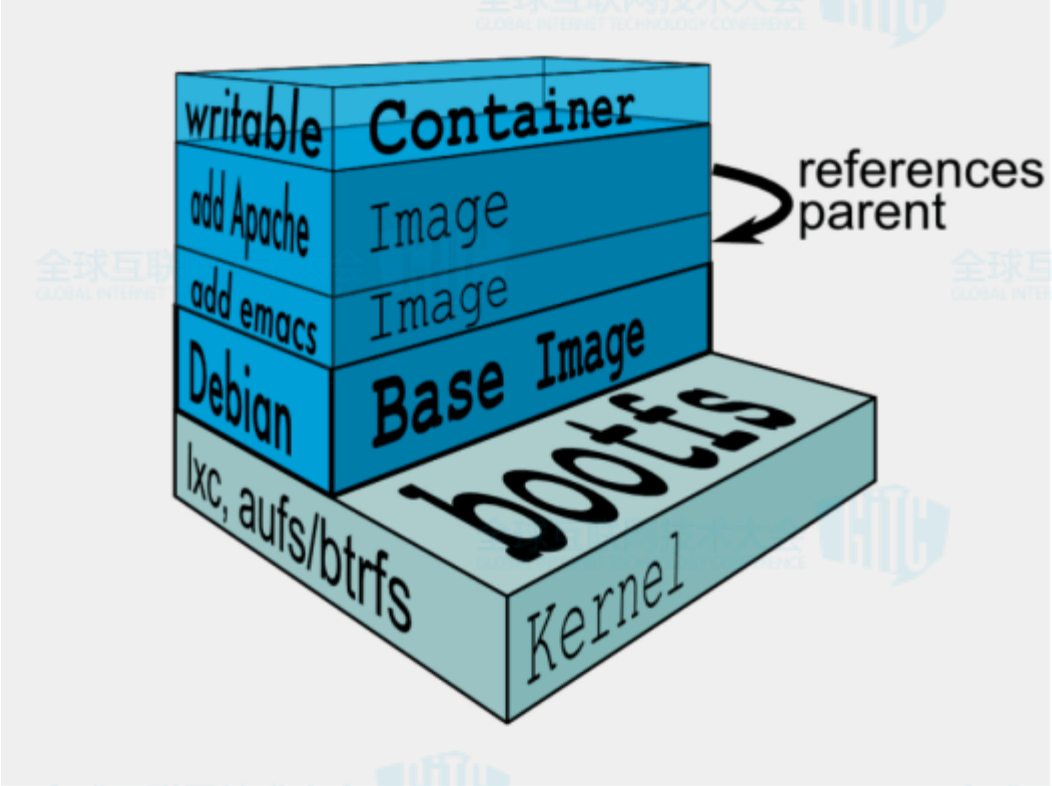
Docker在实际环境中的应用



非生产环境中保证镜像安全可信 生产环境中保证镜像正确的运行

保证Docker镜像的安全

镜像文件简介



Docker镜像是由文件系统叠加而成。最底层是bootfs，之上的部分为rootfs。

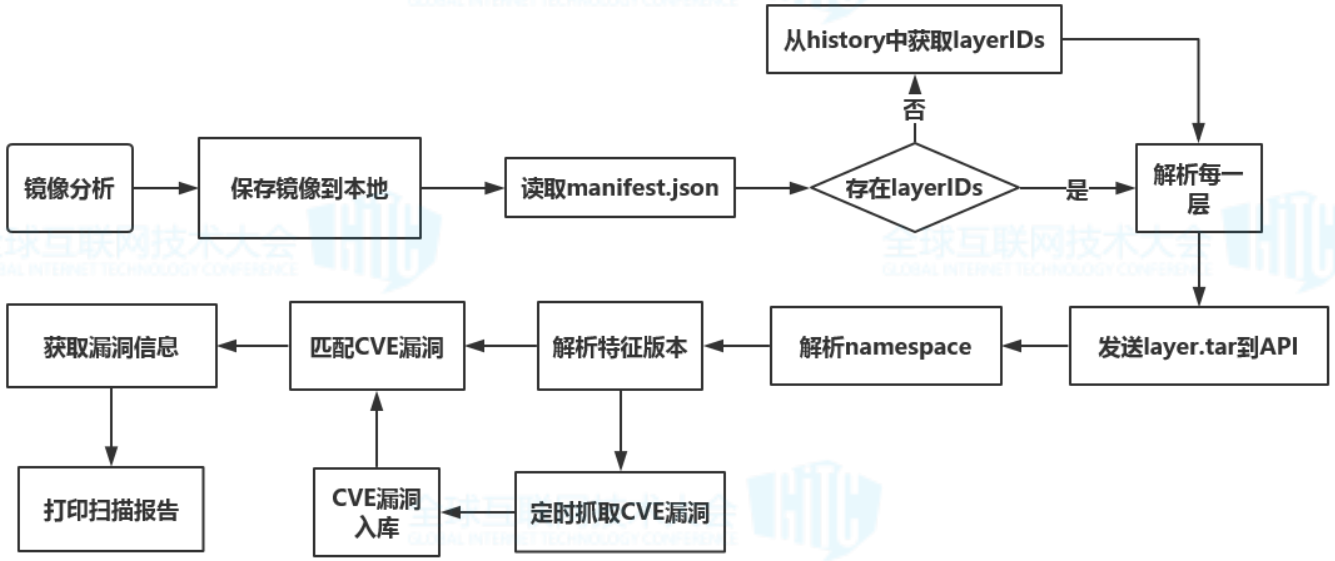
- Bootfs是docker镜像最底层的引导文件系统，包含bootloader和操作系统内核。
- Rootfs通常包含一个操作系统运行所需的文件系统。这一层作为基础镜像。
- 在基础镜像之上，会加入各种镜像，如apache等。

对镜像进行静态安全扫描

Clair扫描结果

```
root@DoSec:~# ./vl nginx
2017/10/26 20:32:54 Saving nginx to local disk (this may take some time)
2017/10/26 20:32:57 Retrieving image history
2017/10/26 20:32:57 Analyzing 3 layers...
2017/10/26 20:32:57 Analyzing 657a3cdde9b0cb24f4b4b7d6c0a9028f58e288092b6983560781af2a8a4b8c
2017/10/26 20:32:57 Analyzing cfd34de5cf3497bee0e6dc67eab48443d1565672e7ad778b92519c14631974
2017/10/26 20:32:57 Analyzing b2946a7c5d3e21b6299eda18aaf8fd108d879e86fd9fe87ebb4e613ff191ab
2017/10/26 20:32:57 Retrieving image's vulnerabilities
Clair report for image nginx (2017-10-26 12:32:57.977540852 +0000 UTC)
CVE-2016-2779 (High)
        runuser in util-linux allows local users to escape to the parent session via
        a crafted TIOCSTI ioctl call, which pushes characters to the terminal's input
        buffer.

        Package:        util-linux @ 2.29.2-1
        Link:           https://security-tracker.debian.org/tracker/CVE-2016-2779
        Layer:          657a3cdde9b0cb24f4b4b7d6c0a9028f58e288092b6983560781af2a8a4b8cfa

CVE-2017-12424 (High)
        In shadow before 4.5, the newusers tool could be made to manipulate internal
        data structures in ways unintended by the authors. Malformed input may lead to
        crashes (with a buffer overflow or other memory corruption) or other unspecified
        behaviors. This crosses a privilege boundary in, for example, certain
        web-hosting environments in which a Control Panel allows an unprivileged user
        account to create subaccounts.

        Package:        shadow @ 1:4.4-4.1
        Link:           https://security-tracker.debian.org/tracker/CVE-2017-12424
        Layer:          657a3cdde9b0cb24f4b4b7d6c0a9028f58e288092b6983560781af2a8a4b8cfa

CVE-2017-8804 (High)
        The xdr_bytes and xdr_string functions in the GNU C Library (aka glibc or libc6)
        2.25 mishandle failures of buffer deserialization, which allows remote attackers
        to cause a denial of service (virtual memory allocation, or memory consumption
        if an overcommit setting is not used) via a crafted UDP packet to port 111, a
        related issue to CVE-2017-8779.

        Package:        glibc @ 2.24-11+deb9u1
        Link:           https://security-tracker.debian.org/tracker/CVE-2017-8804
        Layer:          657a3cdde9b0cb24f4b4b7d6c0a9028f58e288092b6983560781af2a8a4b8cfa

CVE-2017-12944 (Medium)
        The TIFFReadDirEntryArray function in tif_read.c in LibTIFF 4.0.8 mishandles
        memory allocation for short files, which allows remote attackers to cause
        a denial of service (allocation failure and application crash) in the
        TIFFFetchStripThing function in tif_dirread.c during a tiff2pdf invocation.
```

https://github.com/coreos/clair



镜像静态扫描流程

对镜像仅仅进行静态扫描够吗？

对镜像完整的扫描理想状态是

静态检测：

1、安装包版本检测

2、安装包漏洞检查

3、镜像内木马文件检查

4、镜像内病毒检查

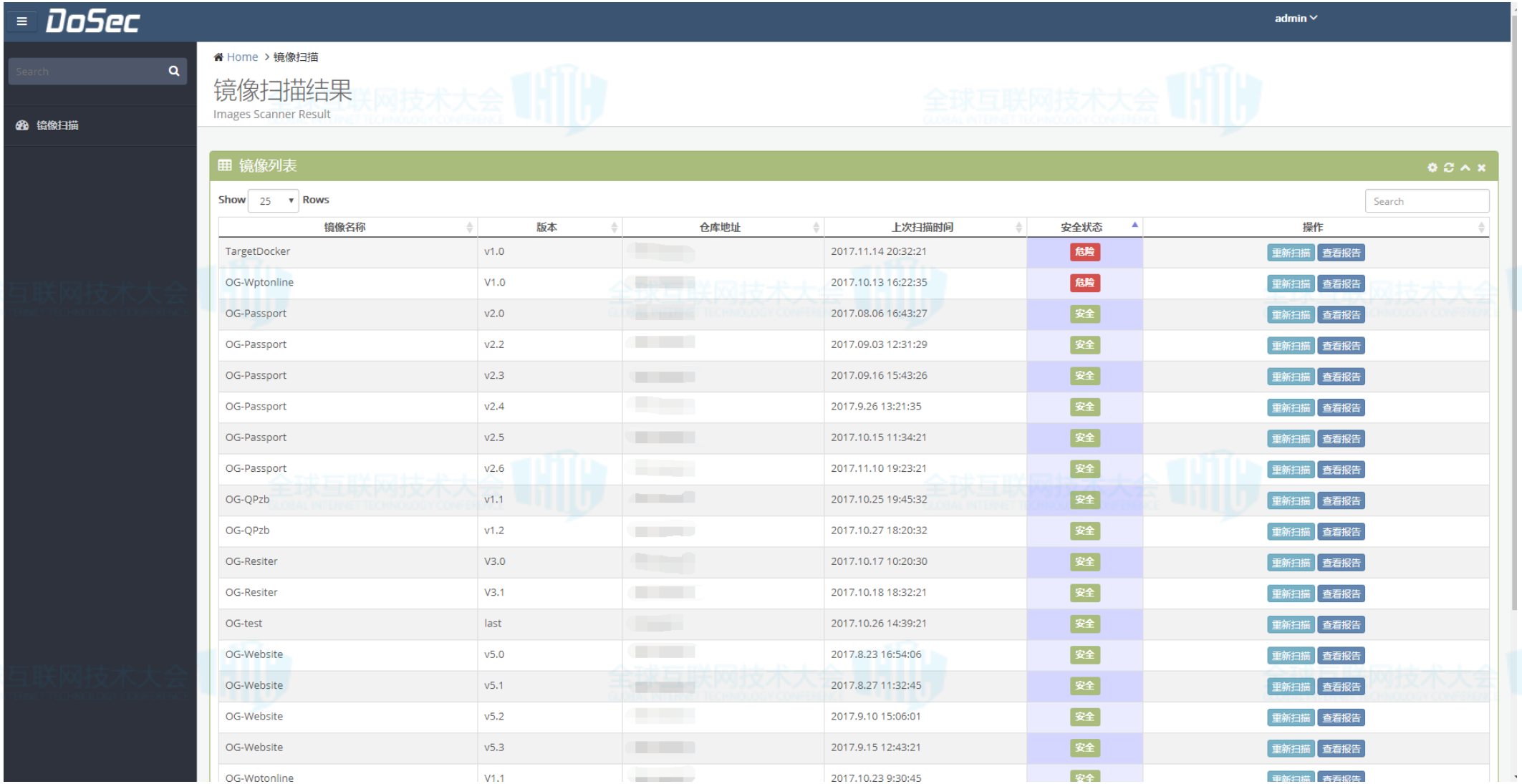
动态检测：

1、镜像系统调用检查

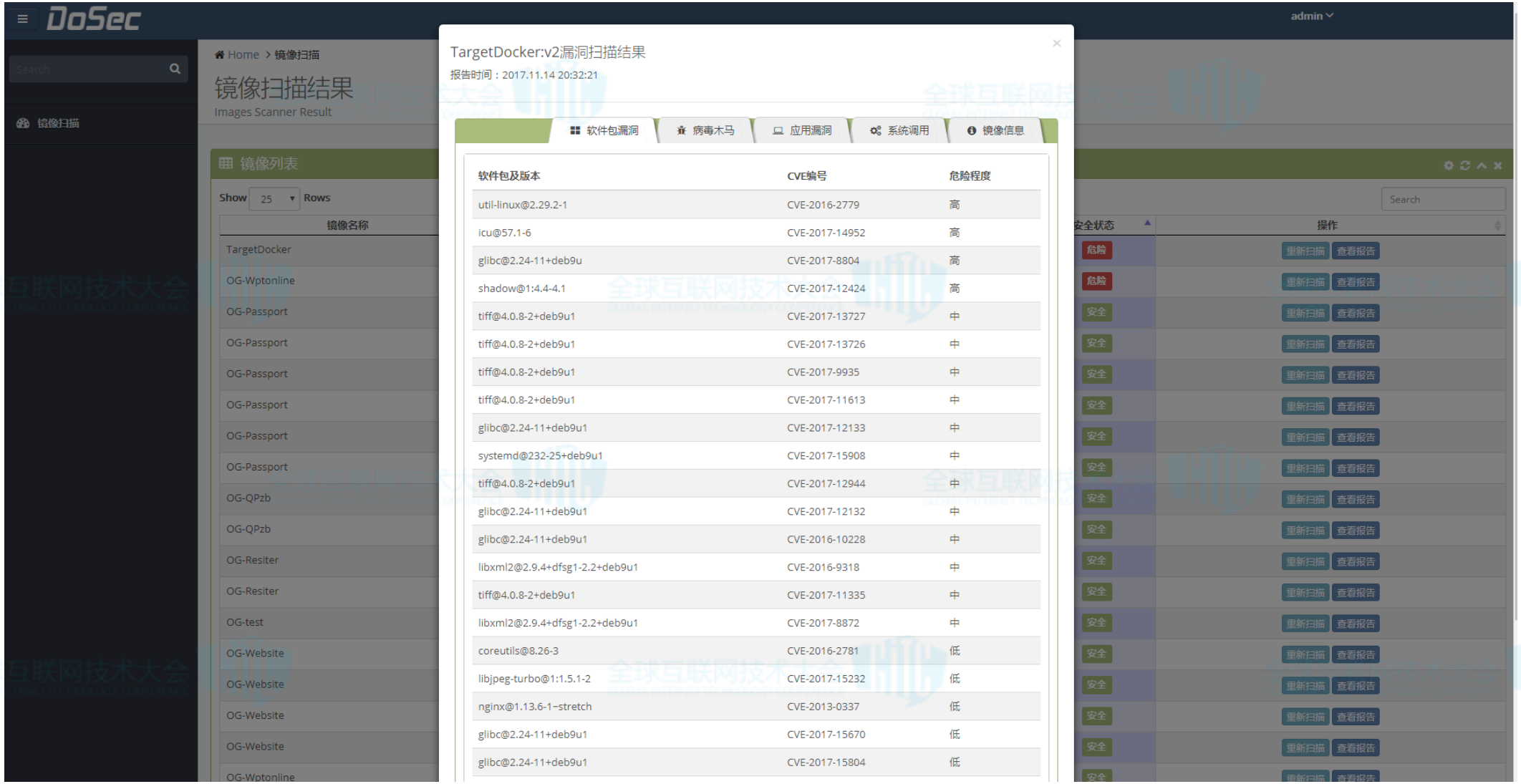
2、镜像网络配置检查

3、镜像内应用安全检查

我们开发一款全面的镜像扫描工具



扫描结果展示一


DoSec
admin
Home > 镜像扫描
镜像扫描结果
Images Scanner Result
镜像扫描
镜像列表
Show 25 Rows
镜像名称
TargetDocker
OG-Wptonline
OG-Passport
OG-Passport
OG-Passport
OG-Passport
OG-Passport
OG-QPzb
OG-QPzb
OG-Resiter
OG-Resiter
OG-test
OG-Website
OG-Website
OG-Website
OG-Website
OG-Wptonline
Search
安全状态
操作
危险
危险
安全
重新扫描
查看报告
TargetDocker:v2漏洞扫描结果
报告时间：2017.11.14 20:32:21
软件包漏洞
病毒木马
应用漏洞
系统调用
镜像信息
软件包及版本
CVE编号
危险程度
util-linux@2.29.2-1 CVE-2016-2779 高
icu@57.1-6 CVE-2017-14952 高
glibc@2.24-11+deb9u CVE-2017-8804 高
shadow@1:4.4-4.1 CVE-2017-12424 高
tiff@4.0.8-2+deb9u1 CVE-2017-13727 中
tiff@4.0.8-2+deb9u1 CVE-2017-13726 中
tiff@4.0.8-2+deb9u1 CVE-2017-9935 中
tiff@4.0.8-2+deb9u1 CVE-2017-11613 中
glibc@2.24-11+deb9u1 CVE-2017-12133 中
systemd@232-25+deb9u1 CVE-2017-15908 中
tiff@4.0.8-2+deb9u1 CVE-2017-12944 中
glibc@2.24-11+deb9u1 CVE-2017-12132 中
glibc@2.24-11+deb9u1 CVE-2016-10228 中
libxml2@2.9.4+dfsg1-2.2+deb9u1 CVE-2016-9318 中
tiff@4.0.8-2+deb9u1 CVE-2017-11335 中
libxml2@2.9.4+dfsg1-2.2+deb9u1 CVE-2017-8872 中
coreutils@8.26-3 CVE-2016-2781 低
libjpeg-turbo@1:1.5.1-2 CVE-2017-15232 低
nginx@1.13.6-1~stretch CVE-2013-0337 低
glibc@2.24-11+deb9u1 CVE-2017-15670 低
glibc@2.24-11+deb9u1 CVE-2017-15804 低

扫描结果展示二

TargetDocker:v2漏洞扫描结果

报告时间：2017.11.14 20:32:21

软件包漏洞 | 病毒木马 | 应用漏洞 | 系统调用 | 镜像信息

名称	类型	位置	危害程度
b374k-2.8.php	webshell	/var/www/html/news.php	高危
silic.php	webshell	/var/www/html/news_index.php	高危
ObfuscatedPhp	webshell	/var/www/html/cmd.php	高危
DodgyPhp	webshell	/var/www/html/manager.php	警告
HiddenInAFile	webshell	/var/www/html/upload.php	提示

关闭

TargetDocker:v2漏洞扫描结果

报告时间：2017.11.14 20:32:21

软件包漏洞 | 病毒木马 | 应用漏洞 | 系统调用 | 镜像信息

系统调用情况：

ExposedPorts:80

PATH:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

NGINX_VERSION: : 1.13.6-1~stretch

NJS_VERSION : 1.13.6.0.1.14-1~stretch

Cmd : "nginx", "-g", "daemon off;

关闭

TargetDocker:v2漏洞扫描结果

报告时间：2017.11.14 20:32:21

软件包漏洞 | 病毒木马 | 应用漏洞 | 系统调用 | 镜像信息

镜像文件分层：

mediaType: application/vnd.docker.image.rootfs.diff.tar.gzip digest: sha256:fa937339c18230d6f834538d120b95590de227d495767cad4d1890d9d9a428c3

mediaType: application/vnd.docker.image.rootfs.diff.tar.gzip digest: sha256:fa937339c18230d6f834538d120b95590de227d495767cad4d1890d9d9a428c3

mediaType: application/vnd.docker.image.rootfs.diff.tar.gzip digest: sha256:fa937339c18230d6f834538d120b95590de227d495767cad4d1890d9d9a428c3

镜像历史：

created:2017-11-04T05:26:48.027090974Z created_by:/bin/sh -c #(nop) ADD file:45233d6b5c9b91e9437065d3e7c332d1c4eb4bce8e1079a4c1af342c450abe67 in / created:2017-11-04T05:26:48.324169027Z created_by:/bin/sh -c #(nop) CMD [\"bash\"] created:2017-11-04T18:40:48.797395609Z created_by:/bin/sh -c #(nop) LABEL maintainer=NGINX Docker Maintainers created:2017-11-04T18:40:48.957138245Z created_by:/bin/sh -c #(nop) ENV NGINX_VERSION=1.13.6-1~stretch created:2017-11-04T18:40:49.130229964Z created_by:/bin/sh -c #(nop) ENV NJS_VERSION=1.13.6.0.1.14-1~stretch created:2017-11-04T18:41:07.163080323Z created_by:/bin/sh -c set -x \t&& apt-get update \t&& apt-get install --no-install-recommends --no-install-suggests -y gnupg1 \t&& \tNGINX_GPGKEY=573BFD6B3D8FBC641079A6ABABF5BD827BD9BF62; \tfound=''; \tfor server in \t\tha.pool.sks-keyservers.net \t\thkp://keyserver.ubuntu.com:80 \t\thkp://p80.pool.sks-keyservers.net:80 \t\tpgp.mit.edu\t; do \t\techo \"Fetching GPG key $NGINX_GPGKEY from $server\"; \t\tapt-key adv --keyserver \"$server\" --keyserver-options timeout=10 --recv-keys \"$NGINX_GPGKEY\" && found=yes && break; \tdone; \ttest -z \"$found\" && echo >&2 \"error: failed to fetch GPG key $NGINX_GPGKEY\" && exit 1; \tapt-get remove --purge --auto-remove -y gnupg1 && rm -rf /var/lib/apt/lists/* \t&& dpkgArch=\"$(dpkg --print-architecture)\" \t&& nginxPackages=\" \t\tnginx=${NGINX_VERSION} \t\tnginx-module-xslt=${NGINX_VERSION} \t\tnginx-module-geoip=${NGINX_VERSION} \t\tnginx-module-image-filter=${NGINX_VERSION} \t\tnginx-module-njs=${NJS_VERSION} \t\" \t&& case \"$dpkgArch\" in \t\tamd64|i386) \t\t\techo \"deb http://nginx.org/packages/mainline/debian/ stretch nginx\" >> /etc/apt/sources.list \t\t\t&& apt-get update \t\t\t;; \t\t*) \t\t\techo \"deb-src http://nginx.org/packages/mainline/debian/ stretch nginx\" >> /etc/apt/sources.list \t\t\t\t\t&& tempDir=\"$(mktemp -d)\" \t\t\t&& chmod 777 \"$tempDir\" \t\t\t\t\t\t&& savedAptMark=\"$(apt-mark showmanual)\" \t\t\t\t\t\t&& apt-get update \t\t\t&& apt-get build-dep -y $nginxPackages \t\t\t&& (\t\t\t\tcd \"$tempDir\" \t\t\t\t&& DEB_BUILD_OPTIONS=\"nocheck parallel=$(nproc)\" \t\t\t\t\tapt-get source --compile $nginxPackages \t\t\t) \t\t\t\t\t\t&& apt-mark showmanual | xargs apt-mark auto > /dev/null \t\t\t&& { [-z \"$savedAptMark\"] || apt-mark manual $savedAptMark; } \t\t\t\t\t\t&& ls -lAFh \"$tempDir\" \t\t\t&& (cd \"$tempDir\" && dpkg-scanpackages . > Packages) \t\t\t&& grep '^Package: ' \"$tempDir/Packages\" \t\t\t&& echo \"deb [trusted=yes] file://$tempDir ./\" > /etc/apt/sources.list.d/temp.list \t\t\t&& apt-get -o Acquire::GzipIndexes=false update \t\t\t;; \tesac \t\t&& apt-get install --no-install-recommends --no-install-suggests -y \t\t\t\t\t\t$nginxPackages \t\t\t\t\t\tgettext-base \t&& rm -rf /var/lib/apt/lists/* \t\t&& if [-n \"$tempDir\"]; then \t\tapt-get purge -y --auto-remove \t\t&& rm -rf \"$tempDir\" /etc/apt/sources.list.d/temp.list; \tfi

TargetDocker:v2漏洞扫描结果

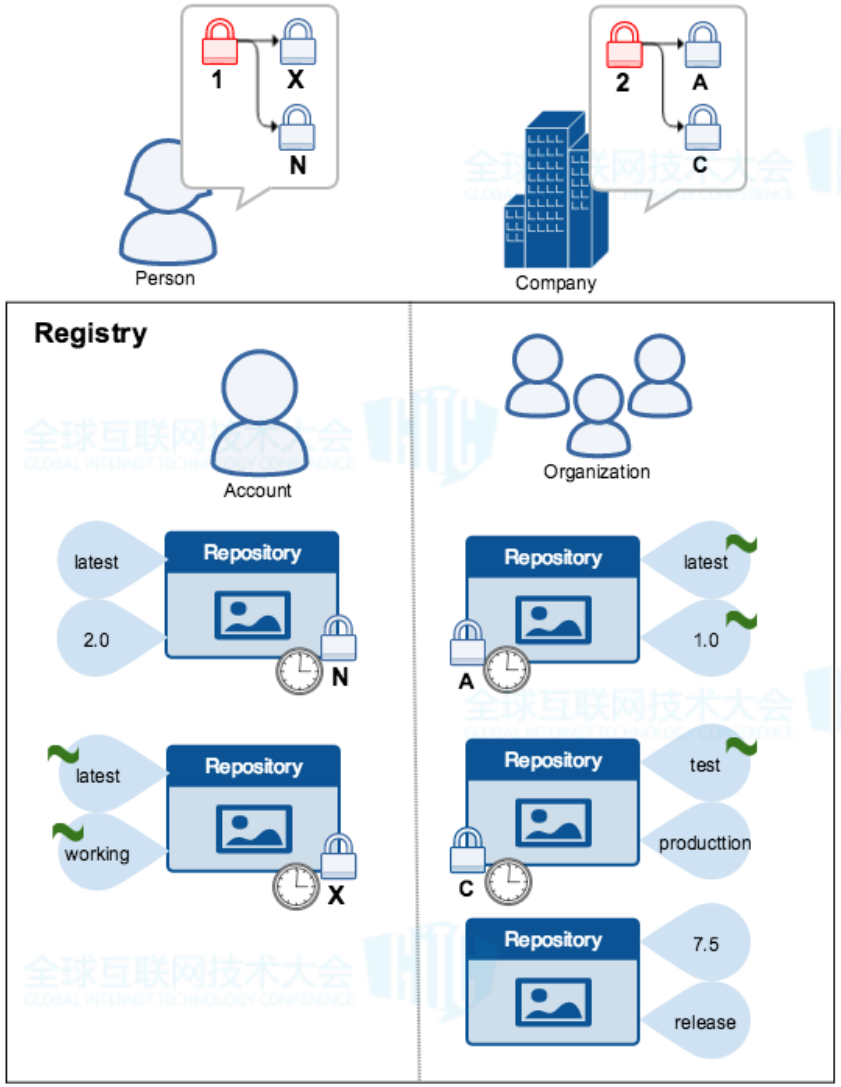
报告时间：2017.11.14 20:32:21

软件包漏洞 | 病毒木马 | 应用漏洞 | 系统调用 | 镜像信息

URL	漏洞描述	危险程度
/cms/news_list.php?id=14	Boolean-based blind SQL injection	高危
/cms/admin/login.php	weak password	高危
/cms/msg/vote.php	Stored - type XSS	高危
/cms/news/list.php	HTML form without CSRF protection	低危

关闭

如何防止不安全镜像进入生产环境




Offline key

A offline key is used to create tagging keys. Offline keys belong to a person or an organization. Resides client-side. You should store these in a safe place and back them up.


Tagging key

A tagging key is associated with an image repository. Creators with this key can push or pull any tag in this repository. This resides on client-side.


Timestamp Key

A timestamp key is associated with an image repository. This is created by Docker and resides on the server.


Signed tag.

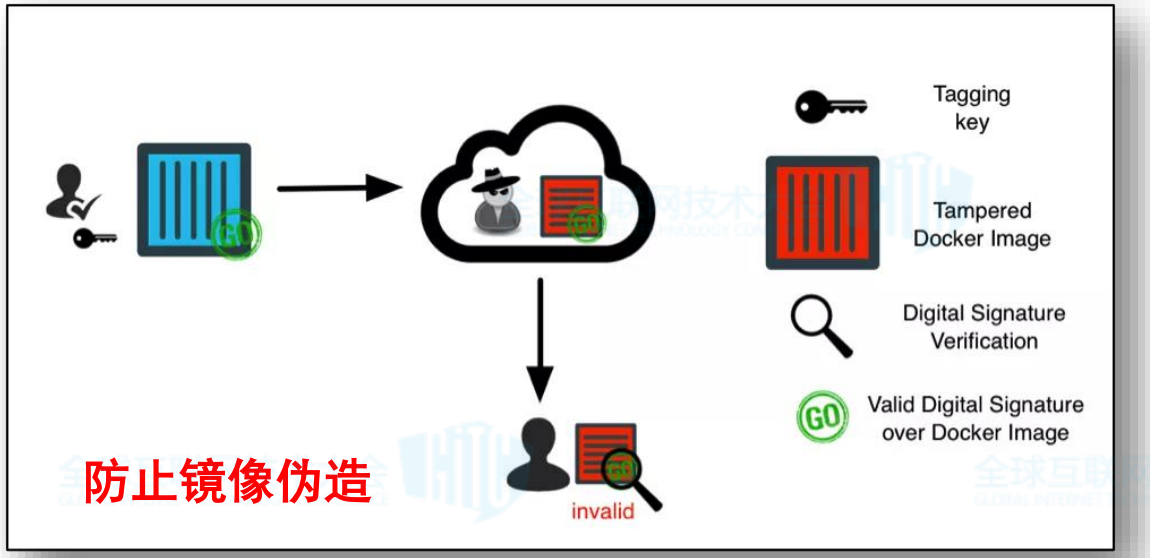
Docker Content Trust

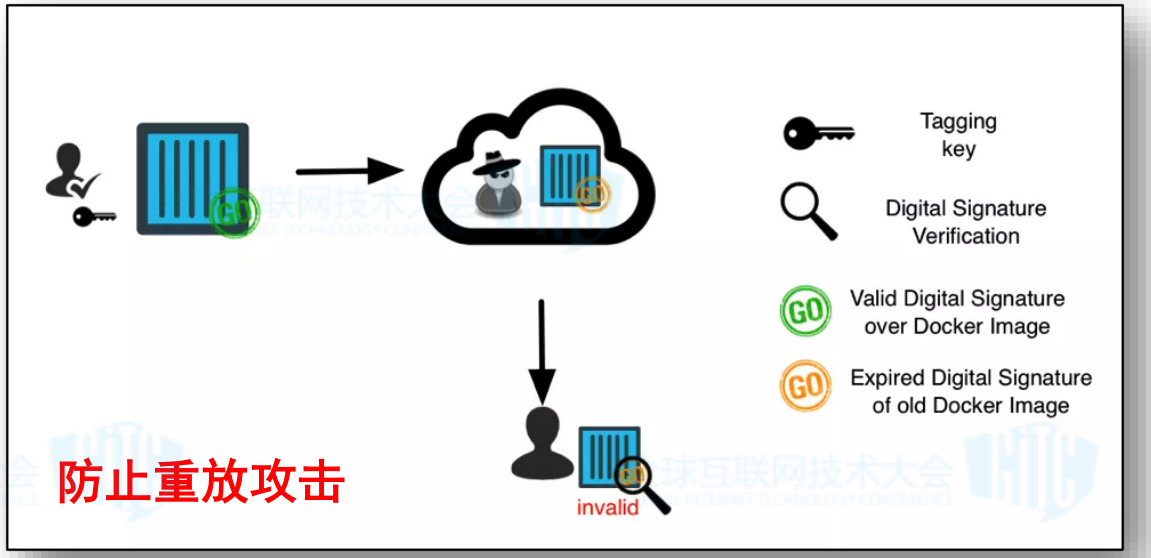
Docker 1.8 版本 后支持

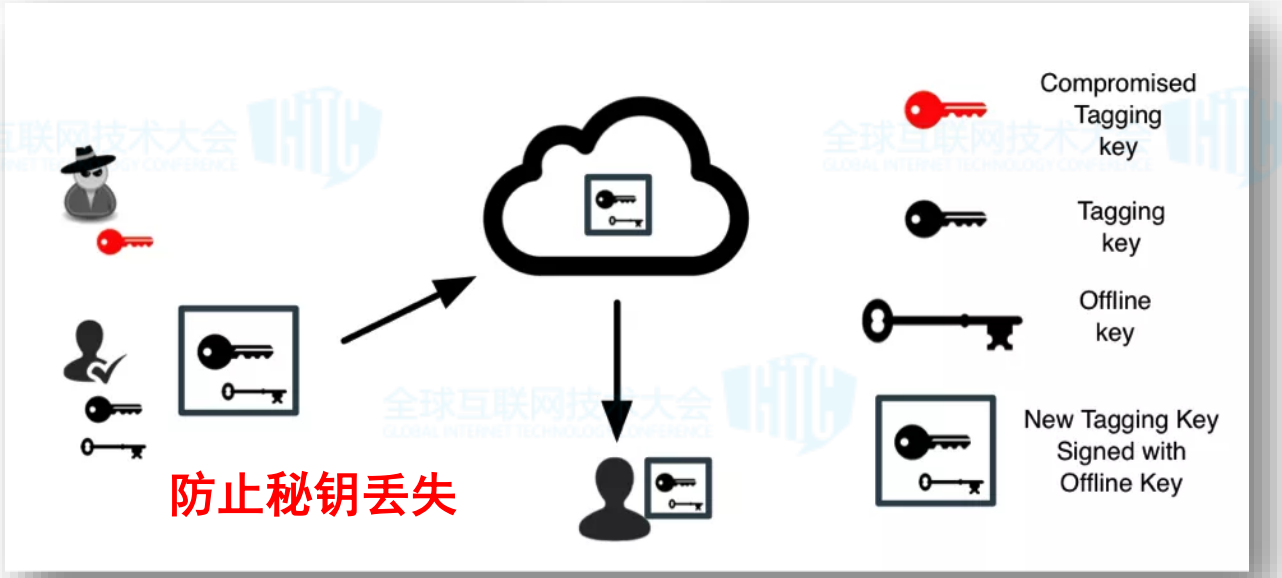
对图像标签的信任是通过使用签名密钥来管理的。首次调用使用内容信任的操作时，会创建一个密钥集。密钥集由以下密钥类组成：

- 作为图像标记内容信任的脱机根密钥
- 仓库或镜像签名标签的密钥
- 服务器时间戳

内容信任的安全效果


Tagging key
Tampered Docker Image
Digital Signature Verification
Valid Digital Signature over Docker Image
invalid
防止镜像伪造


Tagging key
Digital Signature Verification
Valid Digital Signature over Docker Image
Expired Digital Signature of old Docker Image
invalid
防止重放攻击


Compromised Tagging key
Tagging key
Offline key
New Tagging Key Signed with Offline Key
防止秘钥丢失

如何使用内容信任

#export DOCKER_CONTENT_TRUST=1—启用内容信任

export DOCKER_CONTENT_TRUST_SERVER=https://notaryserver:4443—指定验证服务器地址

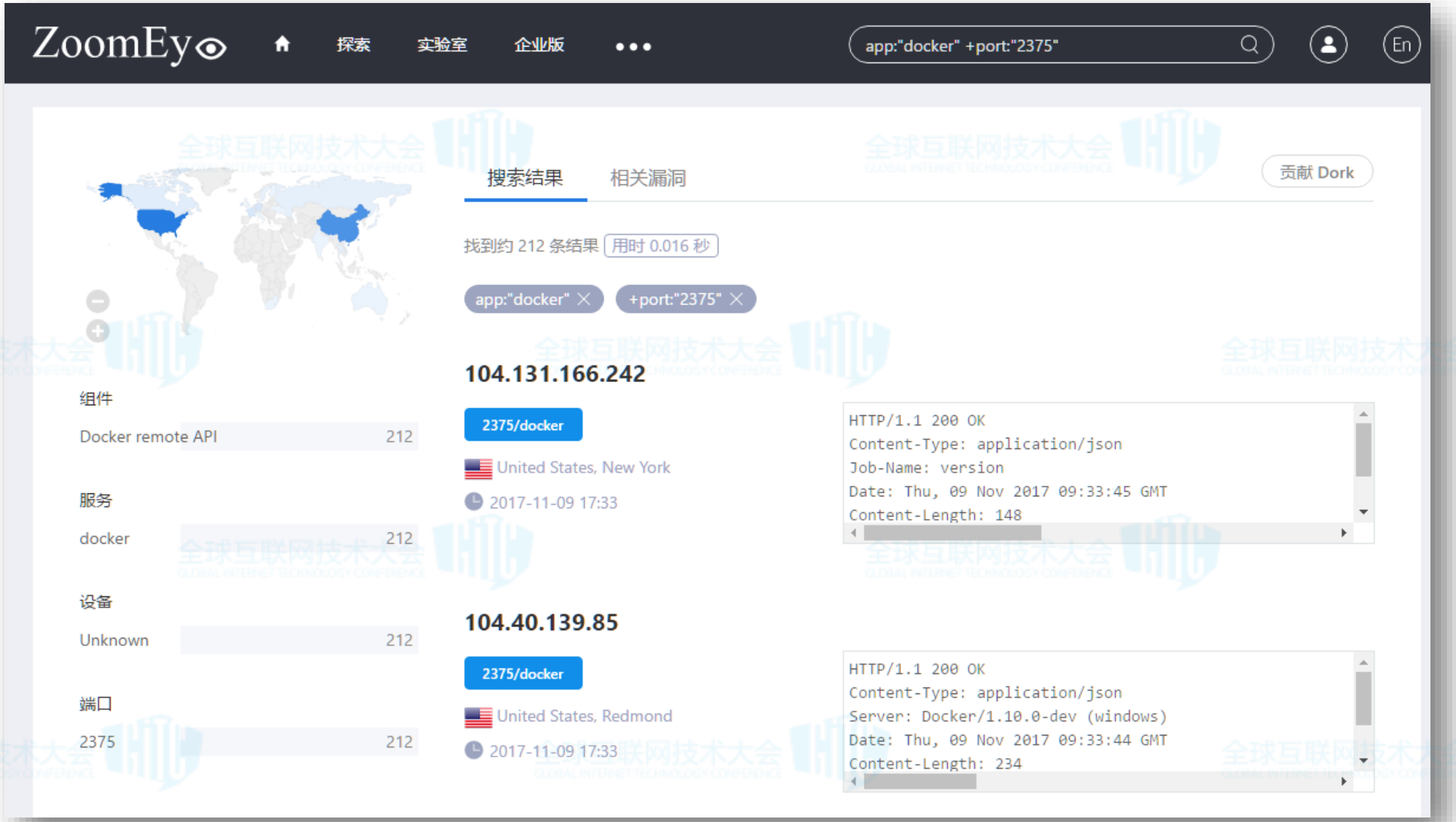
$ docker push <username>/trusttest:testing
The push refers to a repository [docker.io/<username>/trusttest] (len: 1) 9a61b6b1315e: Image already exists 902b87aaaec9: Image already exists latest: digest: sha256:d02adacee0ac7a5be140adb94fa1dae64f4e71a68696e7f8e7cbf9db8dd49418 size: 3220
Signing and pushing trust metadata You are about to create a new root signing key passphrase. This passphrase will be used to protect the most sensitive key in your signing system. Please choose a long, complex passphrase and be careful to keep the password and the key file itself secure and backed up. It is highly recommended that you use a password manager to generate the passphrase and keep it safe. There will be no way to recover this key. You can find the key in your config directory.
Enter passphrase for new root key with id a1d96fb:
Repeat passphrase for new root key with id a1d96fb:
Enter passphrase for new repository key with id docker.io/<username>/trusttest (3a932f1):
Repeat passphrase for new repository key with id docker.io/<username>/trusttest (3a932f1):
Finished initializing "docker.io/<username>/trusttest"

首次推送生成的根密钥

Notary是docker一个官方内容公证服务器项目，利用TUF实现对内容的可信验证
GIT:https://github.com/theupdateframework/notary

保证Docker的接口安全

接口暴露带来的安全问题



启用TLS双向认证

ca.pem	CA证书
Server-cert.pem	服务端证书
Server-key.pem	服务端私钥
Clinet-cert.pem	客户端证书
Clinet-key.pem	客户端私钥

服务端上启动可信连接端口

```
$ dockerd --tlsverify --tlscacert=ca.pem --tlscert=server-cert.pem --tlskey=server-key.pem \ -H=0.0.0.0:2376
```

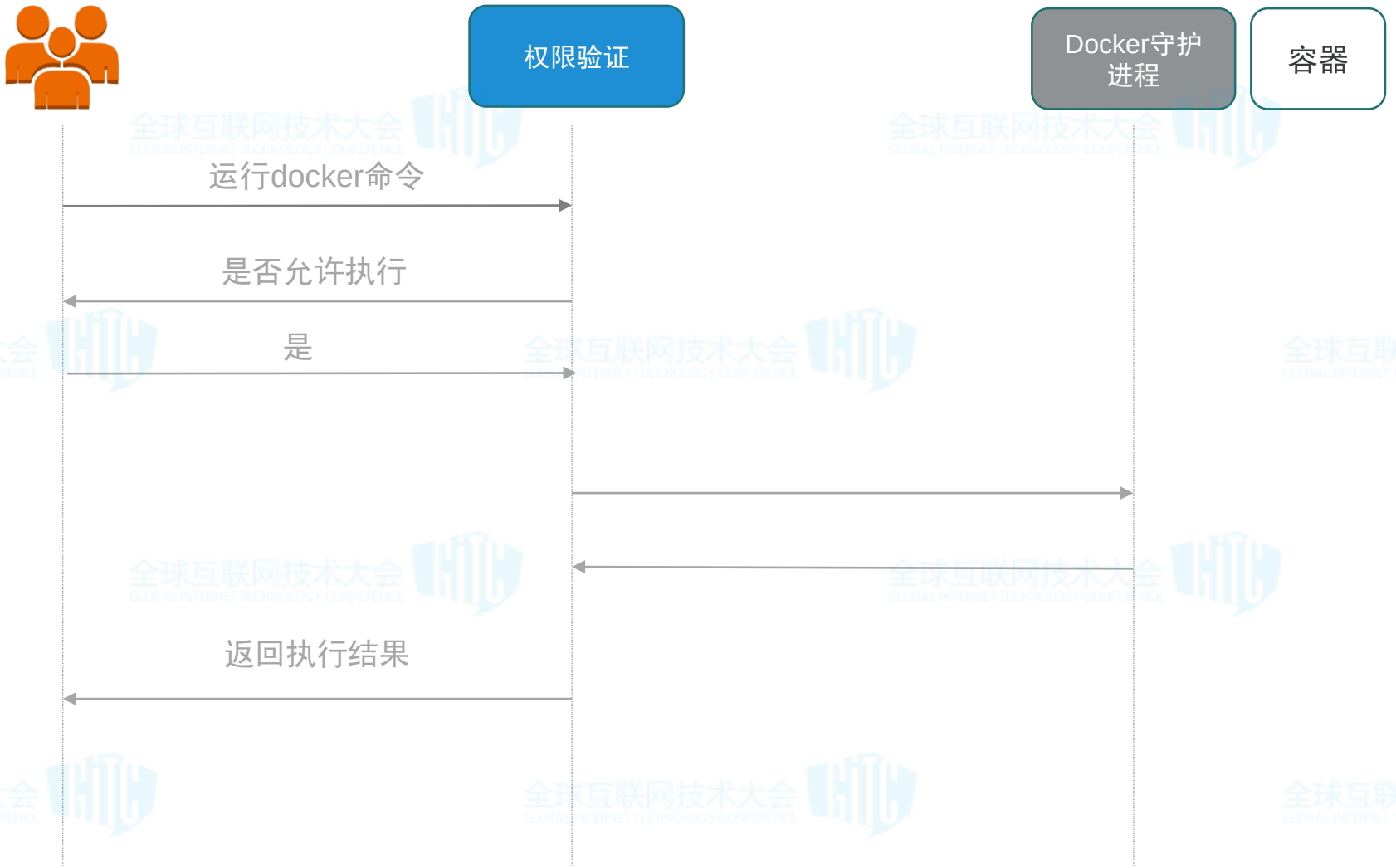
客户端上带证书访问

```
$ docker --tlsverify --tlscacert=ca.pem --tlscert=cert.pem --tlskey=key.pem \ -H=dosec.io:2376 version
```

接口带证书访问

```
$ curl https://dosec.io:2376/images/json \
--cert ~/.docker/cert.pem \
--key ~/.docker/key.pem \
--cacert ~/.docker/ca.pem
```

更细粒度的用户访问控制



利用 Docker 插件实现细粒度控制

Docker 引擎允许用户使用第三方插件的形式扩展 Docker 功能。Docker 的插件类型分为以下三种大类：

•Network plugins 网络插件可以提供容器间互联网络模型。

•Volume plugins 数据卷插件可以使 Docker 数据卷跨多个主机。

•Authorization plugins 验证插件可以提供基于权限的访问控制，也是本文主要讲的插件，比较出名的就是

Twistlock AuthZ Broker 。

Twistlock AuthZ安装及配置

安装插件　https://github.com/twistlock/authz

Twistlock AuthZ Broker 可以在容器中直接安装也可以在Docker外的主机中安装。

$ docker run -d --restart=always -v /var/lib/authz-broker/policy.json:/var/lib/authz-broker/policy.json -v /run/docker/plugins/:/run/docker/plugins twistlock/authz-broker

修改 Docker 启动配置

sudo systemctl edit --full docker.service

```
# add plugin flag ExecStart=/usr/bin/dockerd --authorization-plugin=authz-broker
```

授权策略

在路径 /var/lib/authz-broker/policy.json 下配置授权内容

{"name":"policy_1","users":["alice"],"actions":[""]}

```
root@ubuntu:~/apps/miasm# cat /var/lib/authz-broker/policy.json
{"name":"policy_3","users":["client1"],"actions":["container"]}
{"name":"loglolo3","users":["name1"],"actions":["container_list"]}
{"name":"every","users":[""],"actions":["containner"]}
```

此处配置指明方案 policy_1 的用户 alice 可以执行的命令 actions 为所有命令。

{"name":"policy_3","users":["alice","bob"],"actions":["docker_creat"]}

方案 policy_3 用户 alice 和 bob 可以执行的命令只有创建容器。

使用效果

执行 docker version 命令

权限允许结果

```
docker version
Client: Version: 1.12.1
API version: 1.24
Go version: go1.6.3
Git commit: 23cf638 Built:
OS/Arch: linux/amd64

Server: Version: 1.12.1
API version: 1.24
Go version: go1.6.3
Git commit: 23cf638
Built: OS/Arch: linux/amd64</pre>

#日志输出: Sep 04 15:08:29 mj authz-broker[28646]:
{"allow":true,"err":"","fields.msg":"action
'docker_version'
```

权限拒绝结果

```
Client: Version: 17.03.1-ce
API version: 1.27
Go version: go1.7.5
Git commit: c6d412e
Built: Mon Mar 27 17:14:09 2017
OS/Arch: linux/amd64
Error response from daemon: authorization denied by
plugin authz-broker: no policy applied (user: '' action:
'docker_version')
```

执行结果被拒绝，提示没有相应的方案，权限验证被拒绝。

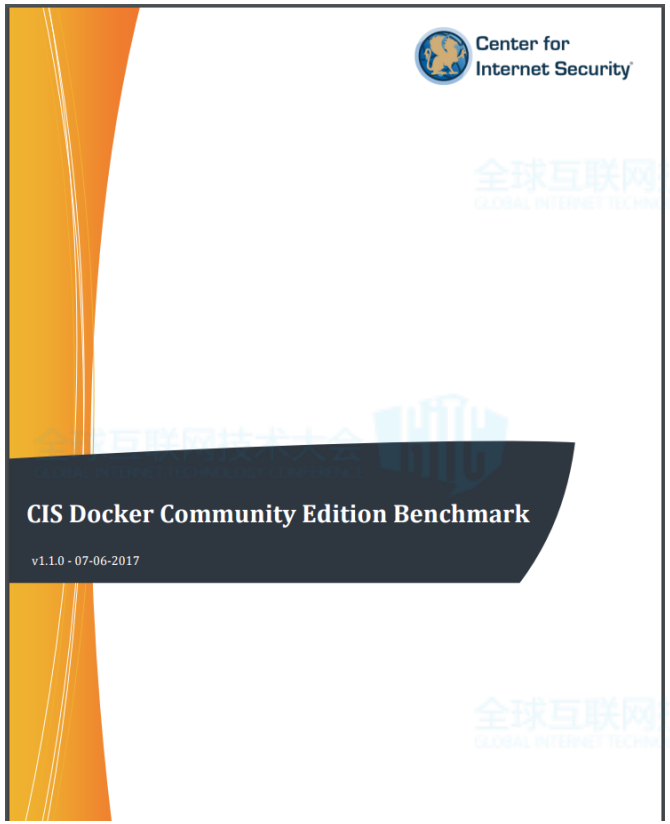
Docker运行及环境安全配置

举例一些安全配置方法

- 容器使用非root用户运行
- 使用安全的基础镜像
- 删除镜像中的setuid和setgid权限
- 最小安装原则
- 对docker宿主机进行安全加固
- 限制容器之间的网络流量
- 启用用户命名空间支持
- 限制容器的内存使用量
- 适当设置容器CPU优先级

.....

Docker安全最佳实践-CIS



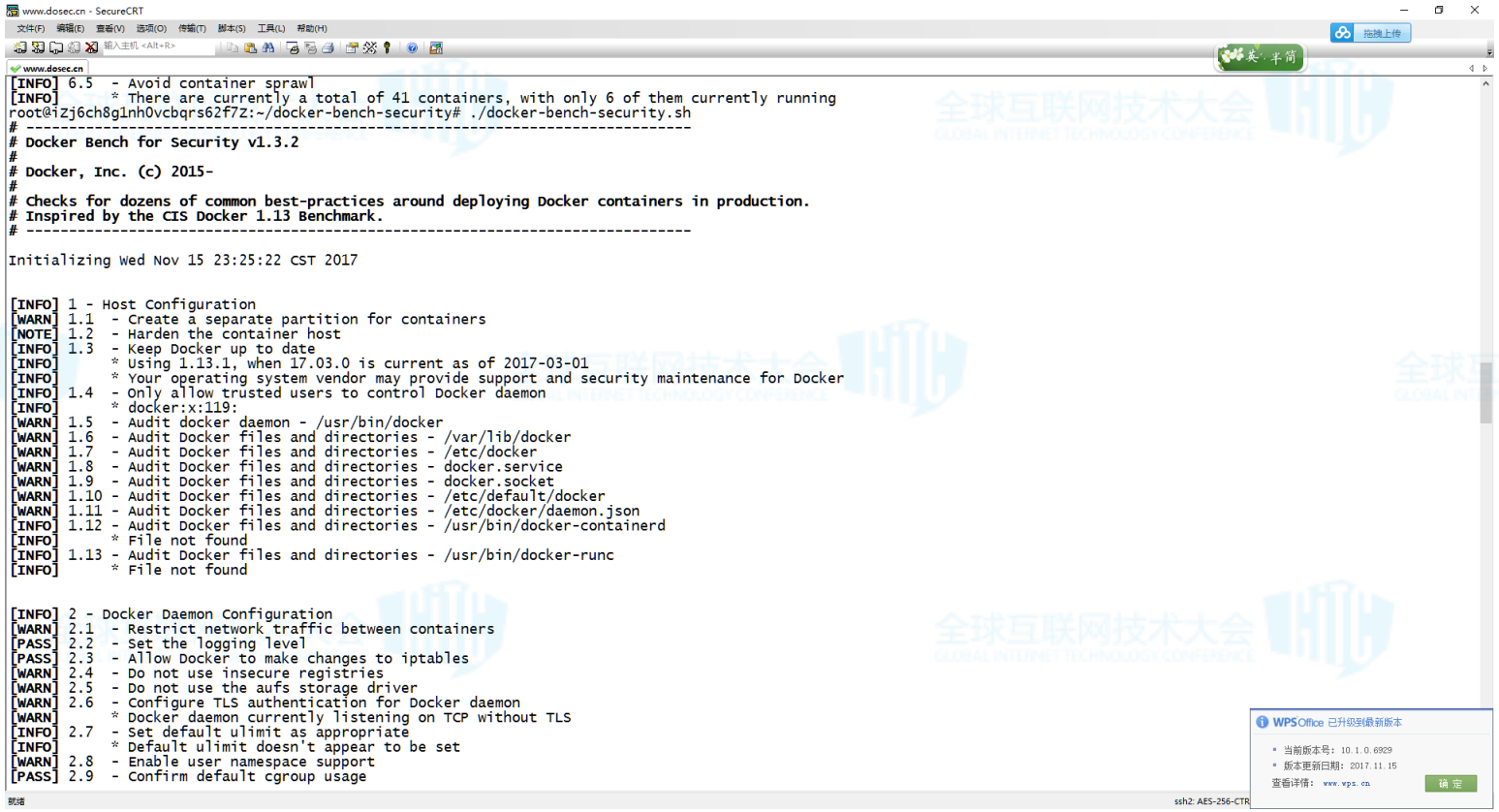
Docker安全配置基线最终版 - Microsoft Excel

	类别	检查项	适用对象	描述	理由	审计方法	判断方法	补救措施
5	1-主机安全配置	1.4 从主机中删除所有非必需服务 (不计分)	Level 1 - Linux Host OS	主机上只运行必要的服务器	最小安装原则	检查主机，确保它专门用于运行Docker。不能找到其他服务。其他服务包括Web服务器，数据库等除容器主进程之外的服务	ps -ef ss -nlp netstat -nlp rpm -qa (or equivalent)	做好禁用掉其他服务
6		1.5 使Docker保持最新 (不计分)	Level 1 - Linux Host OS	Docker容器解决方案正在迅速发展成熟和稳定。像任何其他软件一样，供应商发布Docker软件的定期更新，解决安全漏洞，产品错误并引入新功能。保持这些产品更新，并在新的安全漏洞被修复时升级。	通过了解Docker更新的情况，修复Docker软件的漏洞。攻击者可能会尝试利用已知漏洞攻击，不安装Docker更新可能会让您更容易受到攻击	查看Dockker的最新版本	docker version	去官方下载安装Docker最新版本
7		1.6 仅允许受信任的用户控制Docker守护进程 (计分项)	Level 1 - Linux Host OS	Docker守护进程当前需 root"权限。添加到 docker"组的用户可以获得完整的 root"权限。	Docker允许您在Docker主机和访客容器之间共享目录，而不会限制容器的访问权限。这意味着您可以启动容器并将主机上的/目录映射到容器。然后，容器将能够更改您的主机文件系统，而不受任何限制。简单来说，这意味着您可以通过 docker"组的成员的权限，在主机上启动具有映射/目录的容器。	在Docker主机上执行以下命令，并确保只有受信任的用户是 docker"组的成员。	getent group docker	从 docker"组中删除不受信任的任何用户。另外，不要在主机到容器卷上创建敏感目录的映射。
8		1.7 审计docker守护进程 (计分项)	Level 1 - Linux Host OS	审计所有活动的Docker守护进程	除了审核您的常规Linux文件系统和系统调用外，还可以审核Docker守护程序。Docker守护程序以 rbot"权限运行。因此，有必要审核其活动和使用情况。	验证是否有针对Docker守护程序的审核规则。例如，执行以下命令：	auditctl -l | grep /usr/bin/docker 这应该列出Docker守护程序的规则	为Docker守护进程添加规则 比如： 例如， 在/etc/audit/audit.rules文件中将下面的行添加到下面的行中： -w / usr / bin / docker -k docker 然后，重新启动审计守护程序。 service auditd restart
9		1.8 审核Docker文件和目录 - /var/lib/docker (Scored)	Level 1 - Linux Host OS	审计以下目录： /var/lib/docker.	除了正常的Linux文件系统和系统调用审核外，还可以审核所有与Docker相关的文件和目录。Docker守护程序以 rbot"权限运行。它的行为取决于一些关键文件和目录。/ var / lib / docker就是一个这样的目录。它包含有关容器的所有信息。必须经过审计。	验证是否有与/ var / lib / docker目录相对应的审核规则。	可以执行以下命令查看： auditctl -l | grep /var/lib/docker	为/var/lib/docker 目录添加审计策略。For example, 在/etc/audit/audit.rules 文件中添加如下行： -w /var/lib/docker -k docker 然后重启审计进程： service auditd restart
10		1.9 审核Docker文件和目录 - /etc/docker (Scored)	Level 1 - Linux Host OS	审计以下目录 /etc/docker	除了正常的Linux文件系统和系统调用审核外，还可以审核所有与Docker相关的文件和目录。Docker守护程序以 rbot"权限运行。它的行为取决于一些关键文件和目录。/ etc / docker是一个这样的目录。它具有用于Docker守护程序和Docker客户端之间的TLS通信的各种证书和密钥。必须经过审计。	验证是否有与/etc/docker 目录相对应的审核规则。	可以执行以下命令查看： auditctl -l | grep /etc/docker	为/etc/docker 目录添加审计： 在/etc/audit/audit.rules 文件中添加如下行： -w /etc/docker -k docker 然后重启审计进程： service auditd restart
11		1.10 审计 Docker文件和目录 - docker.service (计分项)	Level 1 - Linux Host OS	审核docker.service（如果适用）。	除了正常的Linux文件系统和系统调用审核外，还可以审核所有与Docker相关的文件和目录。Docker守护程序以 rbot"权限运行。它的行为取决于一些关键文件和目录。docker.service是一个这样的文件。如果管理员更改了守护程序参数，则docker.service文件可能会出现。它支持Docker守护进程的各种参数。如果适用，它必须经过审计。	第一步，找出文件位置： systemctl show -p FragmentPath docker.service Step 2: 如果该文件不存在，则此建议不适用。如果文件存在，请验证是否存在与文件相对应的审核规则： auditctl -l | grep docker.service		如果文件存在，为其添加审计策略： 在/etc/audit/audit.rules 文件添加一行： -w /usr/lib/systemd/system/docker.service -k docker 然后重启审计进程： service auditd restart

Docker公司与美国互联网安全中心（CIS）合作,制定了docker的最佳安全实践，其中包括了主机安全配置、docker守护进程配置、docker守护程序配置文件、容器镜像和构建、容器运行安全、docker安全操作六大项，99个控制点。几乎覆盖了docker安全要求各个方面，我们团队也对其进行了翻译和整理，稍后会免费贡献出来。

CIS安全检查工具

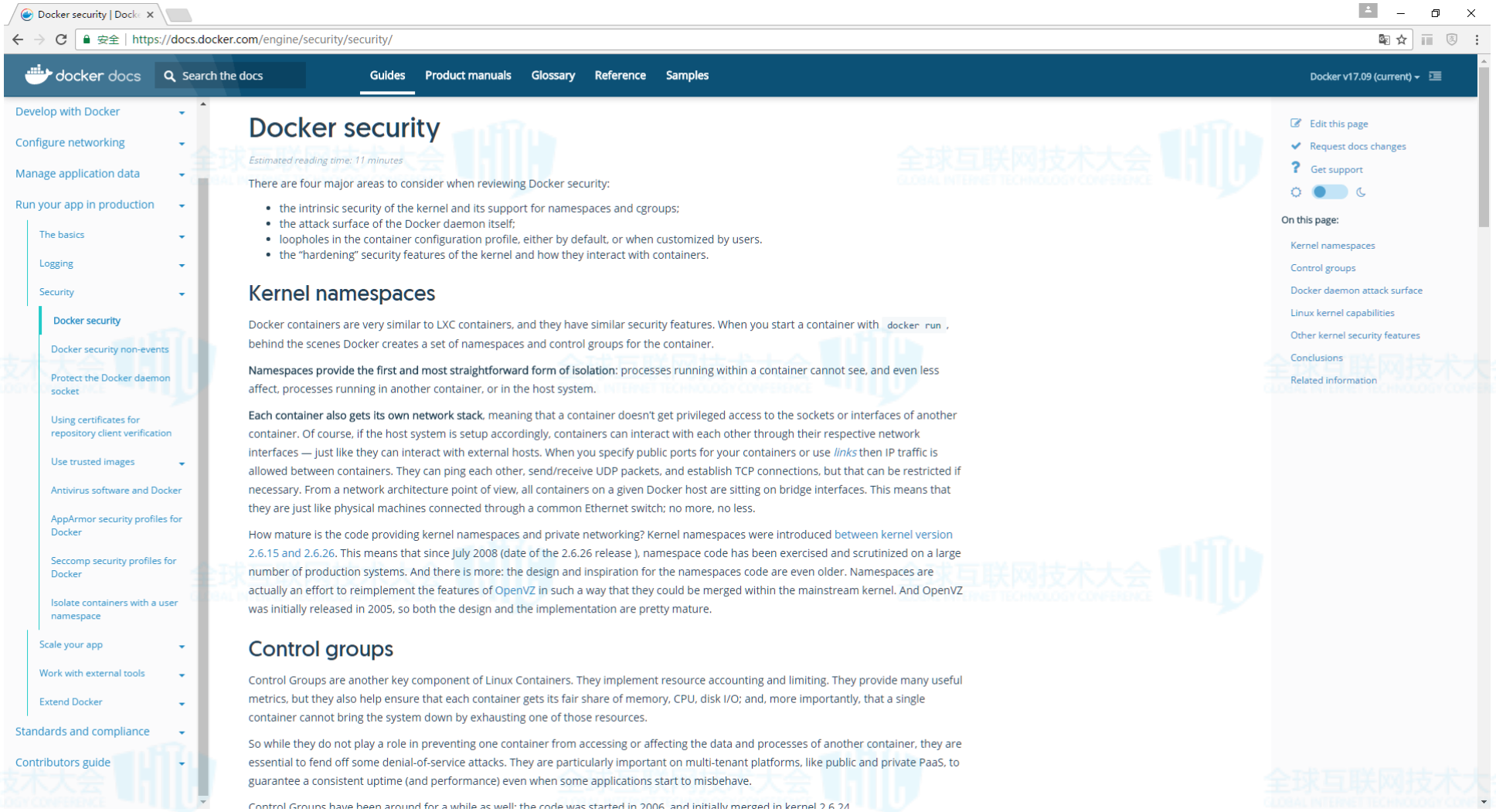
docker-bench-security



https://github.com/docker/docker-bench-security

Docker 安全相关资源

Docker官方文档


Docker security | Docke
安全 | https://docs.docker.com/engine/security/security/
docker docs
Search the docs
Guides
Product manuals
Glossary
Reference
Samples
Docker v17.09 (current)
Develop with Docker
Configure networking
Manage application data
Run your app in production
The basics
Logging
Security
Docker security
Docker security non-events
Protect the Docker daemon socket
Using certificates for repository client verification
Use trusted images
Antivirus software and Docker
AppArmor security profiles for Docker
Seccomp security profiles for Docker
Isolate containers with a user namespace
Scale your app
Work with external tools
Extend Docker
Standards and compliance
Contributors guide
Docker security
Estimated reading time: 11 minutes
There are four major areas to consider when reviewing Docker security:
the intrinsic security of the kernel and its support for namespaces and cgroups;
the attack surface of the Docker daemon itself;
loopholes in the container configuration profile, either by default, or when customized by users.
the "hardening" security features of the kernel and how they interact with containers.
Kernel namespaces
Docker containers are very similar to LXC containers, and they have similar security features. When you start a container with docker run, behind the scenes Docker creates a set of namespaces and control groups for the container.
Namespaces provide the first and most straightforward form of isolation: processes running within a container cannot see, and even less affect, processes running in another container, or in the host system.
Each container also gets its own network stack, meaning that a container doesn't get privileged access to the sockets or interfaces of another container. Of course, if the host system is setup accordingly, containers can interact with each other through their respective network interfaces — just like they can interact with external hosts. When you specify public ports for your containers or use links then IP traffic is allowed between containers. They can ping each other, send/receive UDP packets, and establish TCP connections, but that can be restricted if necessary. From a network architecture point of view, all containers on a given Docker host are sitting on bridge interfaces. This means that they are just like physical machines connected through a common Ethernet switch; no more, no less.
How mature is the code providing kernel namespaces and private networking? Kernel namespaces were introduced between kernel version 2.6.15 and 2.6.26. This means that since July 2008 (date of the 2.6.26 release), namespace code has been exercised and scrutinized on a large number of production systems. And there is more: the design and inspiration for the namespaces code are even older. Namespaces are actually an effort to reimplement the features of OpenVZ in such a way that they could be merged within the mainstream kernel. And OpenVZ was initially released in 2005, so both the design and the implementation are pretty mature.
Control groups
Control Groups are another key component of Linux Containers. They implement resource accounting and limiting. They provide many useful metrics, but they also help ensure that each container gets its fair share of memory, CPU, disk I/O; and, more importantly, that a single container cannot bring the system down by exhausting one of those resources.
So while they do not play a role in preventing one container from accessing or affecting the data and processes of another container, they are essential to fend off some denial-of-service attacks. They are particularly important on multi-tenant platforms, like public and private PaaS, to guarantee a consistent uptime (and performance) even when some applications start to misbehave.
Control Groups have been around for a while as well: the code was started in 2006, and initially merged in kernel 2.6.24
Edit this page
Request docs changes
Get support
On this page:
Kernel namespaces
Control groups
Docker daemon attack surface
Linux kernel capabilities
Other kernel security features
Conclusions
Related information

相关开源项目

- Clair-镜像静态扫描器

https://github.com/coreos/clair

- docker-bench-security-合规检查工具

https://github.com/docker/docker-bench-security

- docker_auth-docker仓库认证服务

https://github.com/cesanta/docker_auth

- Dockerscan-docker渗透工具

https://github.com/cr0hn/dockerscan

- Weave Scope-docker运行监控工具

https://github.com/weaveworks/scope

- Dosec-docker容器相关安全工具和资料

https://github.com/docsec

Thanks!

DoSec公众号

FreeBuf专栏