Security Analysis of Container Images using Cloud Analytics Framework

Byungchul Tak[1] Hyekyung Kim[1] Sahil Suneja[2] Canturk Isci[2] and Prabhakar Kudva[2]

[1] Kyungpook National University, Daegu, Republic of Korea
[2] IBM TJ Watson Research Center, Yorktown Heights, NY, USA

Abstract. Container technology has become an integral part of today's major IT services. Although it offers several benefits, it also introduces new challenges for operating and maintaining secure container environments. One such challenge is to retain the ability to detect and address the containers' vulnerabilities and compliance violations. However, designing an effective solution to enable this capability must be based on the accurate understanding of characteristics observed from actual container images and instances. To contribute toward this objective, we have built a general data processing framework, applying the principles of the state-of-the-art. It is a system that decouples the data collection process from the analysis so as to allow user to focus more on building new analysis logics rather than on the tools for monitoring agents. We applied it to the analysis of container images from the Docker Hub image repository, to learn about their security posture. In this work we present various interesting findings and new insights from analyzing the public image corpus. We have learned that more than 92% of the images contain compliance violations and/or vulnerable packages.

1 Introduction

Container technologies have penetrated deeply into the core of today's major IT services, and its dominance is rapidly expanding. For IT service providers, the adoption of containers is not any more simply one of several options to consider, but rather an imperative that determines the survival within highly competitive market. The main advantages of adopting the container technologies are faster time-to-market, ease of packaging and natural conformity with the dev-ops paradigm. However, such high agility and flexibility coming from these advantages may work against the goal of operating a secure container-based cloud. This is because the speed at which vulnerabilities spread is incomparably faster than that of VMs, due to the ease of access, repackaging and distribution of container images. It has been recently reported that the security vulnerabilities could creep into the container images or container instances without the owners being aware of them, once nested in an image, and it can quickly spread to the wild [26].

The first step towards addressing the security challenges of a container cloud is to gain a clear perspective on the current security postures of containers and images. Although there have been investigations and debates about how secure containers are for production usage [17, 25], we still lack sufficient views on the true nature of container vulnerabilities and their characteristics. Thus, it is crucial that we first build a crisp understanding of the actual security state of containers, to be able to design effective solutions. This would equip users and administrators with knowledge to answer following kind of questions pertaining to containers' security posture: *How many package*

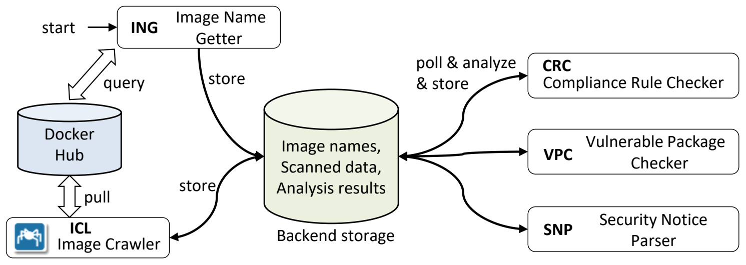

start
ING Image Name Getter
query
store
Docker Hub
pull
ICL Image Crawler
store
Image names, Scanned data, Analysis results
Backend storage
poll & analyze & store
CRC Compliance Rule Checker
VPC Vulnerable Package Checker
SNP Security Notice Parser

Fig. 1. Overall Architecture

vulnerabilities do typical containers have on average, if there are any, and what are the most common vulnerabilities? How conforming public container images are to the well-known best practice rules and what are the most frequently violated ones? How long does it take until we see that previously detected vulnerabilities of container images be eliminated? What impact would it have specifically to my system's security if containers are used?

In order to find answers to aforementioned critical questions, we have conducted an extensive analysis of the top 10,000 public container images from DockerHub [15], one of the most popular container image registry today. We use state-of-the-art introspection tool- Agentless-system-crawler [19, 1], to pull and scan container images to extract system state, such as OS info, file metadata, installed packages, contents of configuration files, amongst others. Extracted state is then pushed to our event-based, and extensible, data processing pipeline, which triggers multiple independent analysis components to operate in parallel, each determining a different aspect of an image's security posture, such as software vulnerabilities, as well as compliance to industry best practices.

In this paper, we first introduce the design and implementation of our data processing pipeline for container security analytics. Then we present the analysis results of the public container images. We find that: *(i)* There is clear trend towards the use of more light-weight images and away from traditional base images, *(ii)* 99% of the images have more than 5 compliance rule violations on average, with password-related ones in high frequency and *(iii)* 92% of the images have 10 vulnerable packages on average.

The rest of the paper is organized as follows. In Section 2, we introduce our data processing pipeline used in analyzing the container images scanning data. Next, we present various security analysis results in Section 3. Then, in Section 4, we provide interpretation and implication of what we have found from the analysis. Section 5 positions our work with existing works. Finally, we conclude in Section 6.

2 Cloud Analytics Framework Architecture

Figure 1 shows the overall architecture of our data analysis platform. It consists of one backend storage that plays a central role and five functional components that interact with it. Functional components, depicted as rounded boxes in the figure, interact only with the backend storage and there is no direct interaction among functional components. They interact with the backend storage by storing data and/or polling new data to consume. For example, VPC (Vulnerable Package Checker) polls for the newly scanned image data to appear in the backend storage. Only when new data appear it initiates the vulnerability analysis. Analysis results are stored back to the backend storage. Appearance of this new analysis results can trigger actions in other functional components which may be polling for it.

Functional components are classified into two groups by how they interact with the backend storage. In the first group they behave strictly as producers. They collect and transform information from outside and insert it into the backend storage so that it would trigger actions on other components. In our architecture, ING (Image Name Getter) and SNP (Security Notice Parser) fall into this category. ING collects image names from the Docker Hub and inserts them to the storage, so that the ICL (Image Crawler) component can determine which images to pull and scan. SNP is responsible for monitoring new security notices of several Linux distros so that it can be used by the VPC component to analyze package vulnerabilities.

Components in the second group behave as both consumers and producers. They look for the appearance of new data in the backend storage through polling, and when it does, they perform their designated data processing. Generated output is fed back into the storage and it can, in turn, be consumed by other components which depend on it. In Figure 1, ICL, VPC, and CRC (Compliance Rule Checker) belong to this second group. Our design allows us to easily extend the analysis capability by adding new functional components as consumers of data produced by existing components. This chaining of components through data dependencies allows higher-order analytics to evolve.

2.1 ING (Image Name Getter)

The Image Name Getter component is responsible for collecting container image names from the Docker Hub. It makes REST calls to retrieve the web pages containing the image lists, and parses out the names and tags. ING has the capability to collect the list of image names in two different styles- *category walk* mode and *sort-by* mode.

- **Category walk:** In this mode, ING first compiles the list of official images by visiting the 'explore' page. It gives us the list of popular official images such as ngnix, httpd, redis and ubuntu. There are about 100 such official images. ING, then, put these as keywords into the Docker Hub search interface. Final list of images and tags are extracted from the result of this search. ING visits the web page of each category in a round-robin manner to compile the list of images from the most popular to the least popular.
- **Sort-by:** In this mode, a user can specify through a REST interface one of the sort conditions. Supported conditions are 'all', 'stars' and 'downloads'. In this study we have used the 'downloads' condition which sorts images by the pull count. Docker Hub does not provide exact pull counts, but categories such as "10M+", "5M+", "1M+", "500K+", "100K+", "50K+" and "10K+".

Each image has one or more 'tags' associated with it, such as 'latest', 'trusty', 'xenial', 'slim', etc., which ING stores as a list with the image name. Although some popular images have a large number of tags, Docker Hub provides only the latest 100 tags.

2.2 ICL (Image Crawler)

The ICL component performs scanning [3] of container images using the Agentless-system-crawler tool (or simply *crawler*). When started, ICL goes into an idle mode

[3] We use the terms 'scan' and 'crawl' interchangeably.

Table 1. Compliance Rules

RULE ID	CATEGORY	DESCRIPTION
A1	SSH	SSH server must not be installed
A2		SSH password-based authentication must be disabled
A3		PermitRootLogin must be disabled
A4		SSH protocol version 2 must be used
B1	Password	Password max age must be 90 days
B2		Minimum password length 8
B3		PASS_MIN_DAYS should be larger than 1
C1-C25	Permissions (file:perm)	(1)/var/log/wtmp:644, (2)/var/run/utmp:644, (3)/etc/shadow:400, (4)/etc/group:644, (5)/etc/passwd:644, (6)/etc/profile:744, (7)/etc/hosts.allow:744, (8)/etc/mtab:700, (9)/etc/sysctl.conf:744, (10)/etc/fstab:644, (11)/etc/sudoers:644, (12)/bin:755, (13)/boot:755, (14)/dev:755, (15)/etc:755, (16)/etc/cron.daily:755, (17)/etc/cron.hourly:755, (18)/etc/cron.monthly:755, (19)/etc/cron.weekly:755, (20)/etc/crontab:644, (21)/etc/ld.so.conf:644, (22)/lib:755, (23)/mnt:755, (24)/root:700, (25)/sbin:755,
D1	Others	Umask must be 022 or more restrictive
D2		Auto log-out after 1 hour
D3		READ/WRITE access of /root/.rhosts must be only by root
D4		READ/WRITE access of /root/.netrc must be only by root
D5		Each UID must be used only once
E1	telnet	telnet server must not exist
E2	rssh	rssh server must not exist
E3	FTP	FTP server must not exist

and waits for the REST call that orders the scanning to start. A user can issue either an auto crawling action or single image crawling via REST interface. In the auto crawling mode, ICL polls the backend storage to check if new image names and tags have been inserted by the ING component. If detected, ICL performs a `docker pull` of the image, and then initiates the crawling action. ICL can be configured to pull, for a given image name, all the images with different tags, or it can be configured to just pull one image that has the most recent tag. We have used the latter since the number of images to crawl can be excessively many with the former option. Crawled output is stored to the backend storage for other components to consume.

The crawler can extract a wide variety of state from a container image. Among the rich set of supported state 'features', we have used files, packages, configurations, and OS features in our platform. The 'files' feature gives us the list of all the files and associated meta-data within the image. The 'packages' feature is the list of installed packages in the image. The 'configurations' feature is the contents of several key configurations files in the image. We use them to determine if the configurations are set correctly according to the policy. Lastly, the 'OS' feature holds information such as distro names and OS version.

2.3 CRC (Compliance Rule Checker)

The CRC component's role is to validate if the images conform to a set of best practice 'rules'. Best practice or compliance rules, are a set of rules that are known to strengthen the security of the system. We have compiled compliance rules from several sources [6, 18, 26], as listed in Table 1. Largely, they are grouped into 5 categories - SSH, password, permission, FTP and 'others'. The 'permission' category specifies the desired security permissions that should be set for critical system files. In order to be considered compliant, a file permission must be equal or stricter than the specified value.

CRC periodically polls the backend storage to check if newly crawled data becomes available by the ICL. If it is, CRC evaluates the compliance rules against the crawled data. As a result of the analysis, CRC produces a json result per rule and sends it to the backend storage. Each json document indicates the rule ID, time of checking, and whether or not the image is compliant to the corresponding rule.

As of now, we have compiled a list of 40 compliance rules, and we continue to increase and evolve our compliance rule set. Current rules are mostly about checking the best practices for Linux-class operating system. However, we plan to add other rules for different distros in the future. We also plan to add other application-specific rules.

2.4 VPC (Vulnerable Package Checker)

The VPC component checks the images for the existence of packages that contain known vulnerabilities. In order to detect vulnerable packages for a container image, VPC takes as input the list of packages which are collected and stored into the backend storage by the crawler, and the list of published security notices for several Linux distributions. Collecting the security notices is the responsibility of the SNP component. VPC simply uses the information to carry out the package version comparison. For each package from the list of installed packages in an image, VPC searches the security notices for any descriptions with matching package names. Then, it compares the version number to determine if the installed one is vulnerable. If not, VPC creates a json entry containing the vulnerable package information and stores it in the backend storage.

2.5 SNP (Security Notice Parser)

SNP is an autonomously running component that periodically scans publicly available security announcements, and extracts and stores the package vulnerability information into one of the index in the backend storage. Security announcements contain descriptions of the vulnerabilities, CVE (Common Vulnerability Exposure) IDs, package name, and the fix version. They are usually made available to public in the form of mailing lists. These are the sources which SNP monitors:

- DSA (Debian Security Announce):
  `https://lists.debian.org/debian-security-announce/`
- USN (Ubuntu Security Notices):
  `https://usn.ubuntu.com/atom.xml`
  `https://lists.ubuntu.com/archives/ubuntu-security-announce/`
- CeSA (CentOS Security Alerts)
  `https://lists.centos.org/pipermail/centos-announce/`

3 Analysis of Docker Container Images

3.1 OS Distro Composition

We first look at the popularity of OS distributions the container images are based-off of. Table 2 lists 21 names we found in the order of their occurrence frequencies across all images and Table 3 shows the average number of files and packages in each. In Table 3 distros with less that 10 occurrences are omitted since average would not be meaningful with too small a population. The three most popular distros turn out to be

Table 2. OS Distro names found in the images sorted by percentages

Rank	OS Distro	Count	Percent
1	debian	3189	31.7%
2	alpine	2896	28.8%
3	ubuntu	2634	26.1%
4	centos	766	7.6%
6	scratch	207	2.1%
5	buildroot	202	2.0%
7	fedora	67	0.7%
8	opensuse	32	0.3%
9	ol(Oracle Linux)	20	0.2%
10	arch	15	0.1%
11	photon	11	0.1%
12	rhel	11	0.1%
13	amzn	7	0.1%
14	sles	6	0.1%
15	gentoo	3	0.0%
16	clear-linux-os	2	0.0%
17	slackware,linuxmint,mageia euleros,kali → appeared once		

Table 3. Average number of files and packages per distros

OS Distro	Avg Files	Avg Packages
debian	26k	275
alpine	8.5k	36
ubuntu	31.3k	317
centos	29.9k	287
scratch	6.4k	0
buildroot	2.2k	0
fedora	26.2k	261
opensuse	31.7k	225
ol(Oracle Linux)	21.1k	196
arch	54k	241
photon	7.2k	62
rhel	10.8k	186

alpine, *debian* and *ubuntu*. Combined, they make up more than 87% of all the container images. We can see that the 'alpine' image, which is relatively a new addition to the Linux distros, is placed at the second position, closely chasing the 'debian' images. It is also noteworthy that well-known distros such as *centos*, *rhel* and *fedora* are underutilized by a big margin. The distro labeled as 'scratch' is an image that does not contain files that can be considered as part of the OS. These are the images created from the empty rootfs and usually contains only the application files. We have also found five distro names that appeared only once in our list. But this does not mean only one image exist in the Docker Hub. There are many derived image from them, but they were not popular enough to be scanned by us. To understand their characteristics, we look at the file counts and package counts next.

3.2 Package and File Counts

Figure 2 shows the relative position of images within the 2-D space spanned by the number of files and the number of packages. The reason behind alpine's popularity is easy to spot from this figure. Compared to the 'ubuntu' distro, 'alpine' has 4 times smaller file counts and 8 times smaller package counts. These observations seem to suggest that alpine may be a better fit towards higher storage efficiency and faster deployment cycles. This widespread adoption of the lightweight alpine runtime, is promising for other runtime minimization approaches in the container ecosystem [14, 16, 23, 20].

Other distros with comparable number of files as alpine are 'vmware photon', 'rhel', and 'mageia'. However, they vary in the number of packages installed inside. While 'alpine' has less than 40 installed packages, 'mageia' has close to 200 packages. The images labeled as 'scratch' contain no packages, and only a small number of files. These custom images seem to have been built by adding only a few binaries, libraries and some configuration files, to the *scratch* base image, which is an empty rootfs, rather than from one of the general purpose OS images, such as 'ubuntu' or 'debian'.

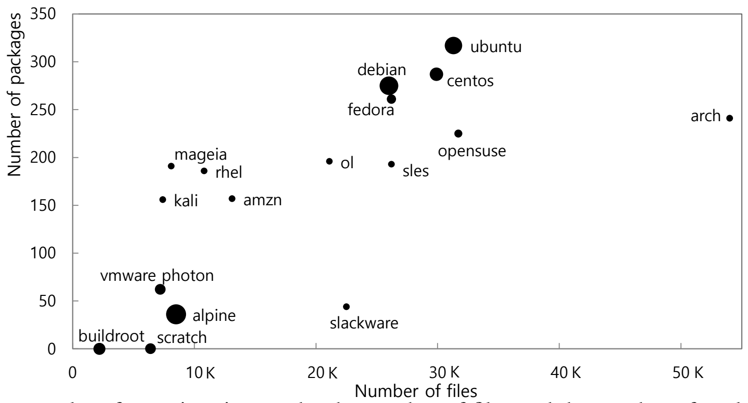


Fig. 2. Scatter plot of container images by the number of files and the number of packages. Dot Size is proportional to the popularity rankings for top 10 distros. Minor distros are all drawn in the same small size to avoid being too tiny.

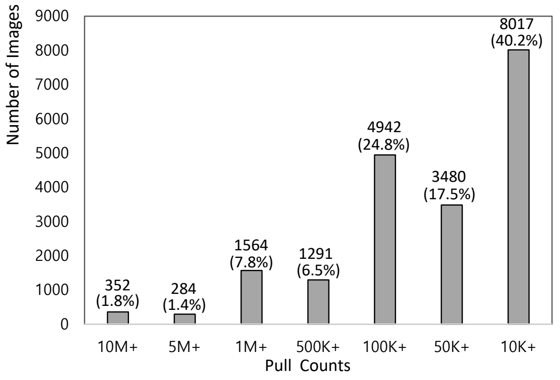


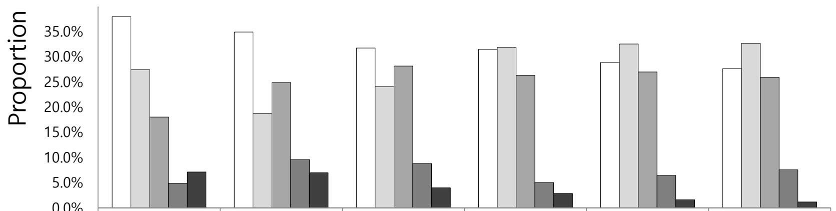
Fig. 3. Number of images grouped by the pull counts.

In terms of the file counts, there are a few distros that have less number of files than 'alpine' -'vmware photon','buildroot','kali', and 'scratch'. Among them, 'kali' and 'buildroot' are both special-purpose Linux distributions. 'Kali' is a suite of penetration testing software, and 'buildroot' is a pack of tools for automating the build process for Linux-based embedded systems. Although these are comparable with each other in the number of files, they vary in the number of packages installed inside.

Figure 2 also reveals a general trend within the space where the number of files and the number of packages are positively correlated, which is expected. However, the correlation is loose. Within a narrow range of the number of files, the package counts can differ as much as 41% (compare debian and sles in Figure 2). Opposite case also exists. For a similar package counts, 'rhel' and 'opensuse' have even larger differences of file counts.

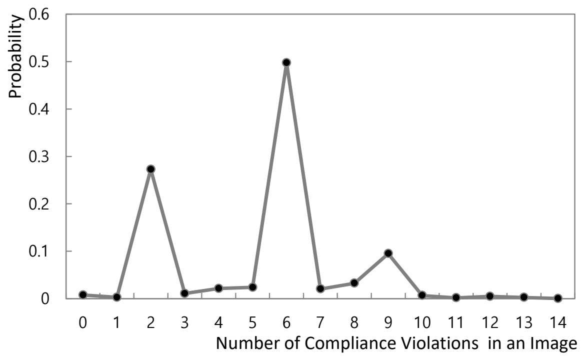
3.3 Image Pull Counts

Docker Hub provides two metrics of images that indicates the popularity - stars and downloads. Stars are the indication of how many users 'liked' the image. Downloads means how many times the image was pulled. We have used the download counts as a measure of popularity in terms of image pulls.



	10M+	5M+	1M+	500K+	100K+	50K+
□ alpine	**38.0%**	**34.9%**	**31.8%**	31.5%	28.9%	27.7%
□ debian	27.4%	18.8%	24.1%	**31.9%**	**32.6%**	**32.7%**
▨ ubuntu	18.0%	24.9%	28.2%	26.3%	27.0%	25.9%
■ centos	4.9%	9.6%	8.8%	5.0%	6.4%	7.6%
■ buildroot	7.1%	7.0%	4.0%	2.9%	1.6%	1.2%

Fig. 4. Top 5 popular images in six pull count groups.



Maximum Number of Violations in an Image	14
Median Violations	6
Average Number of Violations Per Image	5.2
Standard deviation	2.4
Total Number of Images Analyzed	10073
Number of clean images (no compliance violations)	77 (.7%)
Modes (Peaks)	3 (at 2,6,9)

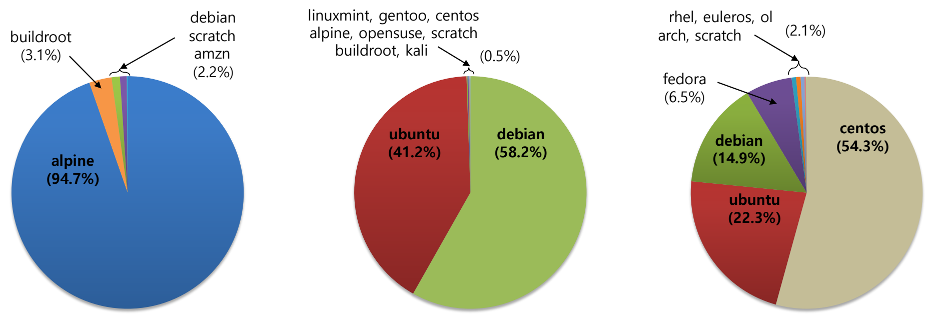
Fig. 5. Distribution of compliance violations and statistics.

Docker Hub provides seven categories of pull counts - 10M+, 5M+, 1M+, 500K+, 100K+, 50K+ and 10K+. Figure 3 shows the number of images per category, as collected by the ING component. Images in the most popular 10M+ group make up only 1.8% of all the images. The proportion of images that have more than one million download counts adds up to be only 11%. Although we have collected image names in all 7 groups, when downloading images for scanning we have not pulled any from the last category, 10K+ because our target of 10,000 images are covered by the first 6 groups.

We have looked at the popularity of distros in each pull count groups and the summary is shown in Figure 4. It shows top five distro names in each group and the corresponding percentages. In all 6 groups, three most popular distros from Table 2 consistently rank as top three. However, the alpine distro is slightly more popular in the first three pull count groups followed by debian. In the rest of the groups (i.e., 500K+, 100K+ and 50K+ groups), debian is ranked as the most popular as highlighted in the Figure 4 with red text. The fact that the proportion of alpine gets higher as the pull count is larger could be reflection of recent increasing popularity of alpine images as the container base.

3.4 Compliance Rule Violations

In this Section we present our analysis results on the compliance violation counts. First, we look at the overall distribution of the number of compliance violations in Figure 5.



(a) Mode at 2 (b) Mode at 6 (c) Mode at 9

Fig. 6. Composition of distros at peaks.

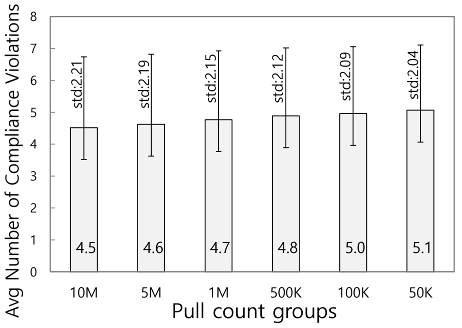
It is a distribution of the average number of compliance violations per container image. Out of 10073 images we have analyzed, only 77 images came out to be free of any compliance violations. On average there were 5.2 compliance violations per image. We suspected that the existence of multiple modes strongly indicated that it was probably a mixture of three distributions. We selected those three peak points (2, 6 and 9 compliance violations on x-axis) and looked at the composition of distros at each point. Figure 6 illustrates the results. At mode 2 the ‘alpine’ image was 94.7%. Thus, the first mode at 2 is mostly due to the ‘alpine’ images having two compliance violations. At mode 3, ‘debian’ and ‘ubuntu’ images were two major components, together taking up 99.5%. We can learn that the number of compliance violations are strongly tied to the distro names. At mode 9, the largest contributing distro was ‘centos’. It is followed by ‘ubuntu’ and ‘debian’.

Table 4 is the list of compliance rules in the order of the violations counts. The count means how many images out of 10073 images have the corresponding rule as a compliance violation. It shows that D2 is violated in almost all the images - 98%. We have set the auto-logout tolerance time to be 3600 seconds, but almost all images had the settings missing from `/etc/profile`. If the setting is not present, we gave a verdict of ‘compliance violation’. For the second commonly violated rule, C3, we have defined the rule to enforce the permission of 400 to `/etc/shadow`. But, many images had the permission of 640, thus, giving us 89.9% violation proportion. The strictness of compliance rules are up to the enforcers. Therefore the list of most violated rules can vary by organization as well as over time. Three password setting rules, B1-B3, are also violated in more than half of the images. Also it is worth mentioning that we have found that ssh server is installed with password authentication enabled in about 7% of the images. This violation can lead to a vulnerability in the containers that can be easily exploited as analyzed in [26]. Out of 25 compliance rules, 8 of them were never violated by any images as listed at the end of Table 4.

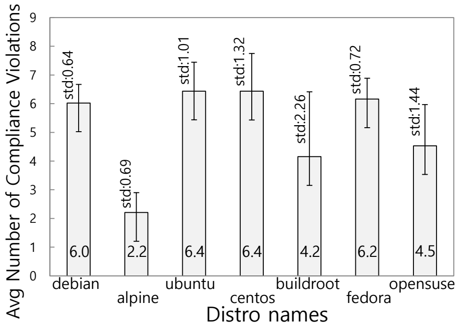
To drill down on how compliance violation behavior changes with image popularity, we grouped the compliance check results by (a) pull counts, and (b) the distro base. Bar graphs in the Figure 7 (a) and (b) illustrate the change of average compliance violations along with the standard deviations in each group. In Figure 7 (a), we see that the average compliance violation increases only slightly as we move from the most popular (10M) to the less popular (50K). The change of average is very small and the standard deviation is also steady across groups. However, we see much larger changes in Figure 7 (b) when

Table 4. List of compliance violations in the order of frequency

ID	Description	Count	%	ID	Description	Count	%
D2	Auto log-out	9869	98.0%	C6	744 /etc/profile	69	0.7%
C3	400 /etc/shadow	9051	89.9%	C12	755 /bin	64	0.6%
B1	Max passwd age	6729	66.8%	C10	644 /etc/fstab	64	0.6%
B3	Passwd min day	6724	66.8%	C25	755 /sbin	51	0.5%
C1	644 /var/log/wtmp	6508	64.6%	C22	755 /lib	33	0.3%
B2	Min passwd len	5811	57.7%	C21	644 /etc/ld.so/conf	22	0.2%
C24	700 /root	1010	10.0%	D5	UID used once	6	0.1%
A3	PermitRootLogin	923	9.2%	C9	744 /etc/sysctl.conf	4	0.0%
D1	UMASK setting 022	894	8.9%	E3	ftp server	3	0.0%
A1	SSH server exists	795	7.9%	C11	644 /etc/sudoers	3	0.0%
A2	SSH passwd auth	673	6.7%	A4	SSH protocol version	3	0.0%
C4	644 /etc/group	155	1.5%	E2	rssh server	1	0.0%
C15	755 /etc	149	1.5%	D4	/root/.netrc	1	0.0%
C2	644 /var/run/utmp	118	1.2%	C17	755 /etc/cron.hourly	1	0.0%
C5	644 /etc/passwd	114	1.1%	C13,C14,C16,C18,C19,C20,C23,D3			none



(a) Avg non-compliance per pull count group



(b) Avg non-compliances by distros

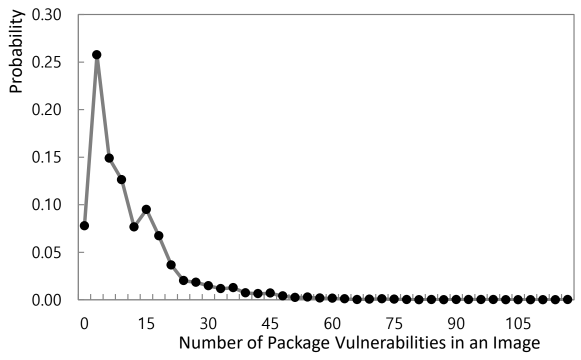
Fig. 7. Compliance Violation Break-down by Pull Count Groups and Distros

we group the images by distros. In accordance with what we observed in Figure 6, the 'alpine' image has the least compliance violations of 2.2 with the second least standard deviation of 0.69. The 'debian' and 'ubuntu' images both have the average around 6, also supporting the reason why we saw the mode at 6 in Figure 5. Although the 'centos' images has similar average of 6.4 as with 'ubuntu' or 'debian', the standard deviation (1.32) is significantly larger than them. This explains why we see the third mode at 9 with 'centos' being the largest contributor.

The reason why average compliance violations do not change much across pull count groups is because each group has roughly equal proportion of 'alpine', 'debian' and 'ubuntu' distros, the top 3 distros as seen in Figure 4. However, since the proportion of the 'alpine' is the largest in 10M+ group among the pull count groups, it has the effect of slightly lowering the average. As we move to less popular groups from 10M to 50K, the proportion of 'alpine' drops, increasing the average from 4.5 to 5.1.

3.5 Package Vulnerabilities

For package vulnerability analysis, we present the results for 'ubuntu', 'debian' and 'centos' images, since we currently consult security notices only for these three distros. We plan to add support for 'alpine' images which use the 'apk' package manager. The total number of images in these 3 distros becomes 6589.



Maximum Number of Vulnerable Packages	116
Median Vulnerabilities	7
Average Number of Vulnerable Packages	10.3
Standard deviation	11.1
Total Number of Images Analyzed	6589
Number of clean images (no vulnerable packages)	512 (8%)

Fig. 8. Distribution of package vulnerabilities and statistics.

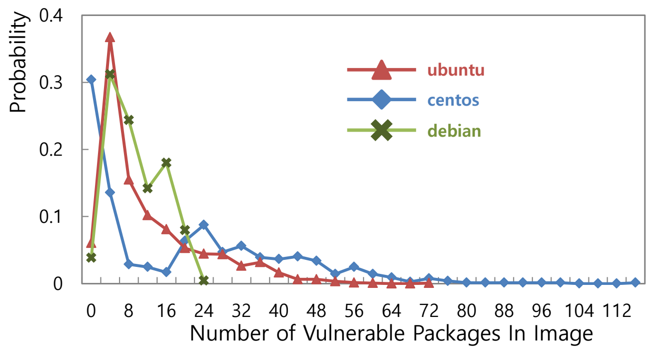


Fig. 9. Component distribution of vulnerable packages.

Figure 8 shows the distribution of package vulnerabilities. Images with zero vulnerabilities turn out to be only about 8% or 512 images. The average package vulnerabilities per image is 10.3, but the difference with the median of 7 is large because the distribution is long-tailed. Small number of images with extremely high vulnerability tend to push up the average. The title of "*the image with the largest number of vulnerable packages*" with 116 vulnerable packages belongs to `gpdejulio/centos:latest`, which was pushed to the Docker Hub about 5 month ago. In fact, all the high ranked vulnerable images are based on the 'centos'. This could be because they were pushed relatively long ago and had not been actively used or updated with recent packages.

Since vulnerabilities are continuously discovered and added, it is natural that the number of vulnerable packages increases as images age. Interestingly, however, 'centos' images had the largest proportion (30%) of clean images. This implies there are subset of 'centos' images are well-cared for while the rest are largely forgotten.

Similar to the distribution of compliance violations, we present three separate component distributions of the vulnerable packages in Figure 9. Stretching to the far right of the distribution, it clearly shows that 'centos' images have much larger number of vulnerable packages in comparison. Out of three distros, 'debian' has the lowest average 7.9 and the lowest max of 24, whereas 'ubuntu' has 10.7 and 72 respectively. 'Centos' has 18.5 and 116. In case of 'debian', there exist two peaks and this indicates that there are largely two groups of debian images with different number of package vulnerabilities.

Table 5. Top 20 Vulnerable Packages

PACKAGE	COUNT	PERCENT	PACKAGE	COUNT	PERCENT
perl	3122	47.4%	git	1424	21.6%
sensible-utils	3007	45.6%	patch	1419	21.5%
openssl	2845	43.2%	gcc-4.9	1129	17.1%
libssl1.0.0	2400	36.4%	apt	1098	16.7%
curl	2385	36.2%	libffi6	962	14.6%
libxml2	2064	31.3%	tar	932	14.1%
libtasn1-6	1947	29.5%	bzr	922	14.0%
gnupg	1926	29.2%	libexpat1	890	13.5%
libgcrypt20	1867	28.3%	ca-certificates	846	12.8%
wget	1835	27.8%	isc-dhcp-client	807	12.2%

Table 6. Top 10 vulnerable packages in three distros

UBUNTU		DEBIAN		CENTOS	
PACKAGE	COUNT	PACKAGE	COUNT	PACKAGE	COUNT
libssl1.0.0	2400	openssl	2750	libstdc++	531
patch	1419	perl	2345	libgcc	526
libffi6	962	sensible-utils	2243	yum	421
libexpat1	890	curl	2064	binutils	421
ca-certificates	846	gnupg	1545	libdb	397
isc-dhcp-client	807	libgcrypt20	1480	kpartx	378
perl	777	libxml2	1461	bind-license	372
sensible-utils	764	libtasn1-6	1397	util-linux	372
systemd	721	wget	1244	libuuid	372
libpam-modules	593	gcc-4.9	1129	libmount	372

Table 5 lists top 20 vulnerable packages we have found. The most frequently seen package in the list is the *perl* package. As of now the most up-to-date stable version for debian is `5.24.1-3+deb9u3`, but we have encountered 16 earlier versions. We further analyzed package vulnerabilities to identify which packages are vulnerable in three distros. Table 6 shows top 10 vulnerable packages in 'ubuntu', 'debian' and 'centos' images. Note that the *perl* is not at the top of 'ubuntu' and 'debian'. For both 'ubuntu' and 'debian', the first one is ssl-related packages - *libssl1.0.0* and *openssl*. Although they are in the same category, they are counted as separate packages. But, when we combine them, we can see that the ssl is the top most package vulnerability, not *perl*. Except for ssl and perl, we can see that the set of vulnerable packages of different distros do not intersect much.

4 Discussion

Trends of Base Images: We observe that the 'alpine' is quickly rising as the standard image base for containers. It is already one of the top three heavily used distros in the Docker Hub images. But, the proportion of 'alpine' is significantly larger in the group of popular images. In Figure 4, 'alpine' takes up 38% of the images in the 10M+ group. Although we have not performed the analysis of creation time, we conjecture that there will be larger proportion of 'alpine' images in more recently created images. As we can see from Figure 2 it make more sense to adopt 'alpine' images than others, since it is much lighter that other images. The size of images is at least an order of magnitude smaller than two other popular distros - 'debian' and 'ubuntu'. Alpine image is in line with the VM-to-container footprint reduction philosophy, while still being general purpose as others. This momentum will continue and we will probably see much larger proportion of 'alpine' images with time. As a consequence of the rise of 'alpine', we see the gradual decline of popularity for 'debian' and 'ubuntu'.

Compliance Behaviors: According to our compliance rule checking, 'alpine' images came out to be most compliant of top three distros. It it too early to conclude that 'alpine' images are safer than others because the compliance rules we have may not contain all the rules appropriate for 'alpine' images. Rather, the current rule set is more geared toward traditional distros such as 'ubuntu'. This motivates us to investigate deeper into the nature of 'alpine' images and build sensible set of compliance policy for 'alpine'. The general pattern we see in terms of compliance rules is that most of the images have weak password settings such as maximum password age or minimum required password length. When it is combined with SSH server, this can create highly vulnerable images where even rookie attackers can easily gain access to the live containers. Compliance violations differ drastically by the distros, but not by the popularity groups. This is because the group contains almost similar combination of distros.

Vulnerabilities: The different base image distros exhibited different vulnerability patterns, and it was hard to find commonalities. However, we were able to discover that there were noticeable distro-specific differences in the magnitude of vulnerable packages. Out of 'debian', 'ubuntu' and 'centos', 'centos' had the highest number of average vulnerable packages in the images. But, on the contrary, 'centos' also had the largest proportion of clean images that had no vulnerable packages. Previously the Banyan blog reported that 30% of official images in the Docker Hub contained high security vulnerabilities [17]. In the report the top packages with high priority vulnerabilities included `mercurial`, `libtasn1-3`, `openssl`, `libmodule-signature-perl`, `binutils`, `mime-support`, `file`, `bash`, `apt`. In our analysis, we find that *libtasn*, *binutils*, and *openssl* are still ranked high in the top 20 vulnerable packages. In that sense, the threat continues in the container images. This strongly suggests we need to adopt continuous scanning of container images to reduce the risk.

5 Related Work

There have been studies on the Docker ecosystem from several perspectives, such as code evolution [10], software engineering [29] and security [24, 9, 12]. Our analysis is a subcomponent of the general Docker ecosystem security, which focuses specifically on the security posture of the Docker image registry- the Docker Hub. We provide a more comprehensive analysis of public images' security posture, as compared to existing tools [22] and studies [17, 25]. We do so by incorporating several image features, beyond just system-level packages, such as file system properties, configuration parameters, and distribution base, amongst others. Furthermore, our data analytics framework is open-sourced and extensible, allowing custom analytics to be incorporated over the dis-aggregated backed storage, without a re-crawl of the container images. Finally, there exist multiple image scanning solutions, such as Anchore [7], Aqua [8], Clair [11] Tenable/Flawcheck [27], and Twistlock [28], amongst others, we use the Agentless-system-crawler to scan container images because of its ability to perform deep inspection, while being open-sourced and extensible [1, 21].

Relative importance of CVEs to a particular deployment are important as well. Some vulnerabilities pose a high risk to the task the image and environment is intended for, others may not. CVEs or vulnerabilities reported have increased significantly in recent years (7900 in 2017) [5], making it increasingly difficult for teams to address them all in time. Some efforts have been made to highlight the risk associated with the vulnerabilities such as the CVSS score, although these remain heuristic scores at best [3]. Others have tried to impose a triage approach [4] in order to focus the developer attention on a limited number of key vulnerabilities that pose the highest risk to that particular deployment (such as product, environment etc). For example, a tar package vulnerability may be ignored, while an openssl immediately addressed. Related work such as the census project [2, 13] have tried to systematically measure risk factors associated with open source projects based on a variety of factors (size of community in the open source, activity, network exposure and so on). In our work, we present results particular to Docker Hub images, and the results should be seen in the light of other related work that provide additional insight to each of the reported metrics.

6 Conclusion

In this paper, we have analyzed container images from Docker Hub to gain a comprehensive understanding of their security posture, as well as their evolution trends. Using our extensible data analytics framework, we have scanned more than 10,000 images, and discovered that 99% of the public images contain 5 compliance violations, and 92% of them have 10 vulnerable packages on average.

As future work, we plan to expand our compliance rules to cover not only the OS checking rules, but also application-specific rules, such as for MySQL, ngnix, etc. Also, we will add capability to check the package vulnerabilities for the popularly rising ‘alpine’ images, as well as for application-level packages such as ruby gems and python pip packages, amongst others.

Acknowledgement

This research was supported by Kyungpook National University Research Fund, 2017.

References

1. Agentless system crawler. `https://github.com/cloudviz/agentless-system-crawler`.
2. Census Project. `https://www.coreinfrastructure.org/programs/census-project`.
3. Don’t Substitute CVSS for Risk: Scoring System Inflates Importance of CVE-2017-3735. `https://securingtomorrow.mcafee.com/mcafee-labs/dont-substitute-cvss-for-risk-scoring-system-inflates-\\importance-of-cve-2017-3735/`.
4. Record-Breaking Number of Vulnerabilities Disclosed in 2017: Report. `https://www.securityweek.com/record-breaking-number-vulnerabilities-disclosed-2017-report`.
5. Vulnerability QuickView. `https://pages.riskbasedsecurity.com/hubfs/Reports/2017/2017\%20Year\%20End\%20Vulnerability\%20QuickView\%20Report.pdf`.
6. Amazon Inspector. `https://aws.amazon.com/inspector/`.
7. Anchore. Open source tools for container security and compliance. `http://anchore.com`.

8. Aqua. `https://www.aquasec.com/`.
9. T. Bui. Analysis of docker security. *arXiv preprint arXiv:1501.02967*, 2015.
10. J. Cito, G. Schermann, J. E. Wittern, P. Leitner, S. Zumberi, and H. C. Gall. An empirical analysis of the docker container ecosystem on github. In *Proceedings of the 14th International Conference on Mining Software Repositories*, pages 323–333. IEEE Press, 2017.
11. Clair. Automatic container vulnerability and security scanning for appc and docker. `http://coreos.com/clair/`.
12. T. Combe, A. Martin, and R. Di Pietro. To docker or not to docker: A security perspective. *IEEE Cloud Computing*, 3(5):54–62, 2016.
13. David, A. Wheeler and Samir, Khakimov. Open Source Software Projects Needing Security Investments . `https://www.coreinfrastructure.org/sites/cii/files/pages/files/pub_ida_lf_cii_070915.pdf`.
14. Docker-slim. `https://github.com/docker-slim/docker-slim`.
15. Dockerhub. `https://hub.docker.com/`.
16. K. Gschwind, C. Adam, S. Duri, S. Nadgowda, and M. Vukovic. Optimizing service delivery with minimal runtimes. In *Proceedings of the 15th International Conference on Service-Oriented Computing*, 2017.
17. Gummaraju, Jayanth and Desikan,Tarun and Turner, Yoshio. Over 30% of Official Images in Docker Hub Contain High Priority Security Vulnerabilities. `https://www.banyanops.com/blog/analyzing-docker-hub/`.
18. James Doran. Is your docker container secure? ask vulnerability advisor! `https://www.ibm.com/blogs/bluemix/2015/07/vulnerability-advisor/`.
19. R. Koller, C. Isci, S. Suneja, and E. de Lara. Unified monitoring and analytics in the cloud. In *7th USENIX Workshop on Hot Topics in Cloud Computing (HotCloud 15)*, Santa Clara, CA, 2015. USENIX Association.
20. Microcontainers Tiny, Portable Docker Containers. `https://blog.iron.io/microcontainers-tiny-portable-containers/`.
21. F. Oliveira, S. Suneja, S. Nadgowda, P. Nagpurkar, and C. Isci. Opvis: extensible, cross-platform operational visibility and analytics for cloud. In *Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference: Industrial Track*, pages 43–49. ACM, 2017.
22. OpenSCAP oscap-docker. `https://github.com/OpenSCAP/container-compliance`.
23. V. Rastogi, C. Niddodi, S. Mohan, and S. Jha. New directions for container debloating. In *Proceedings of the 2017 Workshop on Forming an Ecosystem Around Software Transformation*, pages 51–56. ACM, 2017.
24. E. Reshetova, J. Karhunen, T. Nyman, and N. Asokan. Security of os-level virtualization technologies. In *Nordic Conference on Secure IT Systems*, pages 77–93. Springer, 2014.
25. R. Shu, X. Gu, and W. Enck. A study of security vulnerabilities on docker hub. In *Proceedings of the Seventh ACM on Conference on Data and Application Security and Privacy*, CODASPY '17, pages 269–280, New York, NY, USA, 2017. ACM.
26. B. Tak, C. Isci, S. Duri, N. Bila, S. Nadgowda, and J. Doran. Understanding security implications of using containers in the cloud. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*, pages 313–319, Santa Clara, CA, 2017. USENIX Association.
27. Tenable Network Security. Nessus vulnerability scanner. `http://www.tenable.com`.
28. Twistlock. `https://www.twistlock.com/`.
29. T. Xu and D. Marinov. Mining container image repositories for software configuration and beyond. *arXiv preprint arXiv:1802.03558*, 2018.