2019 IEEE/ACM 16th International Conference on Mining Software Repositories (MSR)

ConPan: A Tool to Analyze Packages in Software Containers

Ahmed Zerouali[1], Valerio Cosentino[2], Gregorio Robles[3], Jesus M. Gonzalez-Barahona[3] and Tom Mens[1]
University of Mons – Belgium[1], Bitergia – Spain[2] and Universidad Rey Juan Carlos – Spain[3]
ahmed.zerouali@umons.ac.be, valcos@bitergia.com, grex@gsyc.es, jgb@gsyc.es, tom.mens@umons.ac.be

***Abstract*—Deploying software packages and services into containers is a popular software engineering practice that increases portability and reusability. Docker, the most popular containerization technology, helps DevOps practitioners in their daily activities. Despite being successfully and increasingly employed, containers may include buggy and vulnerable packages that put at risk the environments in which the containers have been deployed. Existing quality and security monitoring tools provide only limited support to analyze Docker containers, thus forcing practitioners to perform additional manual work or develop ad-hoc scripts when the analysis goes beyond security purposes. This limitation also affects researchers desiring to empirically study the evolution dynamics of Docker containers and their contained packages. To overcome this limitation, we present *ConPan*, an automated tool to inspect the characteristics of packages in Docker containers, such as their *outdatedness* and other possible flaws (e.g., bugs and security vulnerabilities). *ConPan* comes with a CLI and API, and the analysis results can be presented to the user in a variety of formats.**

***Index Terms*—Containers, vulnerabilities, bugs, outdated software, Docker**

I. Introduction

The software development landscape has changed radically, with the need to develop, release and deploy software ever more rapidly. This has given rise to DevOps techniques such as continuous integration, continuous deployment and continuous monitoring. It also led to tools for software containerization, that improve and facilitate portability, reliability and deployment [1]. A container is a lightweight, stand-alone executable piece of software that includes an entire runtime environment [2]. Thus, a container includes an application *plus* all its dependencies, such as system and third-party packages, libraries, binaries, and configuration files. This promotes modularity and eases reproducibility of the software environment. For these reasons, the popularity of containerization technology has significantly increased during the last years. A 2019 survey by *Red Hat* estimates that the reliance on containers is expected to increase by 89% in the next two years [3].

Docker is one of the most popular containerization technologies, which provides a schema to allow executing multiple applications on a single host. *Images* are at the heart of this schema. They are essentially file systems designed to be composed of *layers*, where each layer represents an instruction to be executed when an instance of the image (i.e., a container) is executed. This layered structure allows images to be composed of other images, simplifying their reuse.

Docker images are available in registries such as Docker Hub, that provide a common place to build, update and share images for different architectures such as Linux and Windows. Linux-based images are the most popular on Docker Hub. They usually include system packages that correspond to the used Linux distribution (e.g., Debian), plus a collection of third-party packages that come from popular package managers like *pip* or *npm* (the default package managers for JavaScript and Python). Despite being largely employed, Linux-based images often contain buggy or vulnerable packages which may have been fixed in more recent package releases, thus exposing the production environment where they have been deployed to potential risks [4], [5].

Docker users can rely on a limited number of tools to scan and monitor their images, primarily for security vulnerabilities. For instance, *Anchore.com* [6] inspects container security using CVE-based security vulnerability reports, while *Dagda* [7] relies on the OWASP [8], Red Hat Oval [9] and the Offensive Security exploit database [10]. Finally, *Snyk* [11], an Open Source security platform, does not only scan and monitor, it also suggests fixes for detected vulnerabilities. Docker Hub provides its own tool, *Docker Trusted Registry* which scans each layer and checks the results against a periodically updated vulnerability database. If practitioners wish to evaluate other aspects than security vulnerabilities, such as evaluating the *outdatedness* or quality issues of a container (e.g., in terms of the bugs [12] and available releases [13] of its system and third-party packages), they are forced to develop ad-hoc scripts which may be time-consuming and error-prone. This limitation also affects empirical research in software container mining [14], [15], [16], [17], [18], requiring scholars to re-invent the wheel because of the need to retrieve and analyse the data by themselves.

To overcome this limitation, we developed *ConPan*, a tool that simplifies the analysis of Docker Hub images and their installed packages. *ConPan* gathers and processed information about security vulnerabilities, bugs and freshness of installed packages, leveraging on different publicly available databases. *ConPan* is an easy-to-use open source tool that: (i) allows to track and retrieve data about packages in containers from multiple sources in an easy and consistent way; (ii) has been designed to be extensible to cover new data sources; and (iii) provides the possibility to output its results to external analysis and/or visualization tools thanks to its flexible output formats (e.g., pandas dataframes [19]).

2574-3864/19/$31.00 ©2019 IEEE
DOI 10.1109/MSR.2019.00089

Fig. 1. Overview of *ConPan*. The user interacts with the tool via either the command line or through its API. Once the tool is initialized, the target Docker image is pulled, the contained packages are extracted and tracked back to the corresponding package managers, and vulnerabilities and other bugs are identified and returned as output (as pandas dataframes, JSON documents and/or charts).

The remainder of the paper is organized as follows: Section II briefly describes the approach underlying *ConPan*. Section III presents *ConPan* in action, while Sections IV and V conclude the paper and report on how the tool can be extended.

II. Overview of *ConPan*

ConPan aims to support both practitioners and researchers desiring to analyse Docker containers. Its goal is to collect and fetch data about software packages that are installed in Docker containers, leaving the tasks of storing and analysing the data to other tools. It can be used either as a command-line tool or as a Python library. Release 1.0.0 of *ConPan* supports the analysis of Debian packages included in Docker images based on the the corresponding Linux distribution.

The overall structure of *ConPan* is summarized in Figure 1. Its core is composed by five tasks, which consists of: (i) pulling and running Docker images; (ii) identifying the installed packages; (iii) tracking them back to their package managers; (iv) searching for their known vulnerability reports or other reported bugs and quality issues; (v) reporting the results in a specific output format. *ConPan* also provides general information about the analysed Docker Hub image, fetched from the Docker Hub registry using its API [1].

To be able to trace back installed packages to their package managers, *ConPan* relies on datasets containing historical information about where and when packages were released. For example, the dataset corresponding to Debian packages is obtained via the Debian archive [20], which contains daily snapshots of all Debian packages from the official and security Debian Snapshot repositories. *ConPan* can easily include other datasets of popular package repositories (e.g., *npm*[2] and PyPI[3]) by relying on freely available services. *ConPan* downloads the latest Debian dataset and, using regular Debian tools (`dpkg -l`) in the container, identifies the installed packages and compares them to the latest releases to assess how outdated they are.

To search for known package vulnerabilities, *ConPan* relies on available datasets from security trackers. In the case of Debian, *ConPan* uses its official security tracker [21]. For each reported vulnerability of a package present in the analyzed image, *ConPan* marks the corresponding package release as vulnerable if the vulnerability is still open, or if the vulnerability has been fixed in a more recent version than the one installed in the image.

To search for bugs in packages contained in the image, *ConPan* uses available bug trackers. In the case of Debian, *ConPan* relies on the Ultimate Debian Database [22], which includes data about various aspects of Debian distributions, including package bug reports. For each reported bug of a package present in the analyzed image, *ConPan* identifies the corresponding package release affected by a given bug, if the specific package version is higher or equal to the version where the bug has been first found. In case the bug report is resolved, *ConPan* verifies if the package version is lower than the one fixing the bug.

III. *ConPan* in Action

This section describes how to install and use *ConPan*, highlighting its main features.

A. Installation

ConPan has been developed and tested mainly on GNU/Linux platforms. It is very likely to work out of the box on any Linux-like (or Unix-like) platform, upon providing the right version of Python available (i.e., 3.x). *ConPan* can be installed using Python's pip package manager. Listing 1 shows how to install *ConPan* from its source code. Further installation instructions can be found on the GitHub repository https://github.com/neglectos/ConPan.

Listing 1. How to download and install *ConPan*

```
# Installation from source code using pip
$ git clone https://github.com/neglectos/ConPan
$ python3 setup.py build
$ python3 setup.py install
# To uninstall
$ pip3 uninstall conpan
```

B. Use

Once installed, *ConPan* can be used through a command-line interface (CLI) or through the API of a Python library. We showcase these two types of executions below.

1) CLI: Using *ConPan* as a CLI does not require much effort. Listing 2 shows how easy is to call *ConPan* on a Docker Hub image. Three parameters are required: i) the type of packages (Debian packages in this case); ii) the Docker image to be analyzed; and iii) the path to the historical package dataset extracted from Debian archive. The latter dataset is provided with the tool. In other cases where *ConPan* relies on online APIs such as the *npm* registry, the data path is not needed.

Listing 2. How to use the *ConPan* CLI

```
# Call ConPan from command line
$ conpan -p debian
         -c <Docker image>:<tag>
         -d path/to/data
```

The output of the execution of Listing 2 would be the general information about the analyzed Docker image, plus

[1] https://docs.docker.com/registry/spec/api/
[2] https://www.npmjs.com/
[3] https://pypistats.org/api/

the number of installed packages, vulnerabilities and bugs. The output also includes three figures showing the proportions of the outdated packages, plus the proportion of vulnerabilities and bugs grouped by their severity.
A concrete example of the output of *ConPan* will be shown in subsection III-C.

2) API: The API of *ConPan* can be accessed from within any Python script with minimal effort, assuming the user knows how to program in Python. Listing 3 shows how to use the *ConPan* API. The *ConPan* module is imported at the beginning of the file, then the package kind, Docker image and path to the historical data (if needed) parameters are set and used in order to call *ConPan*. The generated output consists of one JSON file and four pandas dataframes, which are summarized below.

- *general info*: a JSON containing general information about the analyzed Docker image, such as size, architecture, number of pulls, among others.
- *installed_packages*: a dataframe containing the set of all installed packages.
- *tracked_packages*: a dataframe containing the set of installed packages that are coming from the package manager and were not installed from external sources.
- *vulnerabilities*: a dataframe containing the set of all vulnerabilities with their severity, status, corresponding packages, etc.
- *bugs*: a dataframe containing the set of all bugs with their severity, status, corresponding packages, etc.

Listing 3. How to use *ConPan* as a python library

```
#! /usr/bin/env python3
from conpan.conpan import ConPan
# Parameters
kind = 'debian'
image =  <Docker image name>
dir_data = 'path/to/data/'
# Call ConPan
cp = ConPan(packages=kind,
            image=image,
            dir_data=dir_data)

# Results
(general_info, installed_packages,
tracked_packages,
vulnerabilities, bugs) = cp.analyze()
```

C. Reporting

The output generated by *ConPan* can be exploited directly using *pandas* dataframes, one of the most commonly Python libraries for data analytics [19]. The reported data can be visualized by means of *matplotlib* [23] or *seaborn* [24] libraries, or by converting the dataframes to JSON files and storing them to a persistent NoSQL document-based storage, such as an ElasticSearch database [25], or as raw data in a CSV format. Jupyter notebooks [26] can be used for data analysis and early prototyping of data visualization in a transparent way[4].

[4]Python notebooks allow to store the results and the code needed to produce them

The *ConPan* library can be very useful for researchers desiring to analyze container packages. The tool can be used to extract data about a large number of Docker containers and to create datasets to be used in empirical research. The dataset contains different information related to software packages installed in containers, thus providing a powerful basis to perform empirical studies. As an example, we have used *ConPan* in previous work [27], where we empirically analyzed installed system packages in a large dataset of Docker Hub images that are based on the Debian Linux distribution.

For deployers desiring to monitor their Docker containers, they can use the tool's CLI or integrate its functionalities in the automation of their Docker image builds. For instance, Listing 4 shows the output of *ConPan* executed on the community Docker Hub image *google/mysql*[5], which is an image of MySQL server for Google Compute Engine. The listing includes general information such as the description, the number of pulls and stars, and the last time the image was updated. As can be seen the image was not updated for more than three years,

Listing 4. General information about the Docker image *google/mysql*

```
Results:
General information about the Docker image:  google/
    mysql
- description: MySQL server for Google
               Compute Engine
- star_count: 18
- pull_count: 46692
- full_size: 96687899
- last_updated: 2015-11-13T01:19:18.235940

Results about installed packages in:  google/mysql
# installed packages: 144
# tracked packages: 81
# vulnerabilities: 240
# bugs: 474
```

Figure 2 shows the pie charts generated by *ConPan* to highlight the percentage of outdated packages, vulnerabilities and bugs. We observe that the image has a high proportion of outdated packages (93.8%), and a high number of vulnerabilities (240) and bugs (474). We also observe from the breakdown in severity that most vulnerabilities are medium and high.

IV. Conclusion

Deploying applications and services using containerization technologies is becoming a popular practice in software engineering, thanks also to the increasing popularity of Docker containers. They offer isolation, portability and reusability by providing all needed artifacts and dependencies shipped in one package. However, as shown in previous research [4], [27], [18], Docker containers may contain vulnerable and outdated packages that may put at risk the environments where the containers are deployed. Nevertheless, little support is given to practitioners and researchers desiring to assess the status of their containers, thus forcing them to resort to the tedious task of writing error-prone ad-hoc scripts.

[5]https://hub.docker.com/r/google/mysql

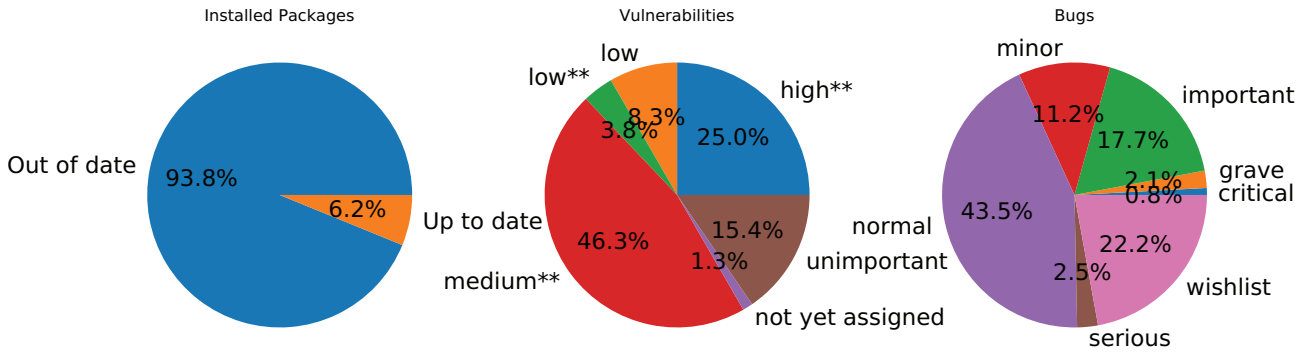

Fig. 2. Statistics about Debian packages in the community Docker Hub image **google/mysql**

For this reason, we presented *ConPan*, a tool that simplifies the monitoring and analysis of software packages installed in Docker containers, by reporting how outdated, vulnerable and buggy they are. *ConPan* can be used stand-alone as a CLI or integrated through its Python API with other processes (e.g., empirical studies or automation of Docker image builds). The output of *ConPan* has been conceived to cover a wide spectrum of technologies such as Python libraries for data processing and visualization (e.g., pandas, matplotlib), NoSQL document-based databases, JSON and CSV formats.

V. Future Work

Besides the software packages of the host operating system, a container may include third-party packages that are needed at runtime or for the development of other included applications. Two of the most popular third-party packages are JavaScript and Python packages that are hosted in the *npm* and PyPI package repositories, respectively. For this reason, we aim to extend *ConPan* to support this kind of packages. This should be easy to achieve, given *ConPan*'s extensible architecture. We have already started to support *npm* packages by relying on basic *npm* commands[6]. After extracting the list of all *npm* installed package versions we can assess how outdated they are with respect to their latest available releases by leveraging on the *npm* registry API[7]. As an example, Figure 3 shows the number of package updates, grouped by type (i.e., major, minor or patch), that *npm* package versions installed in the Docker image *node:stretch* are missing.

For third-party Python packages, we will collect all names and versions of the installed Python packages in the target Docker container. Then, we will use the PyPI registry API to see how outdated they are by comparing them with their latest available releases. To search for any known vulnerabilities of the installed packages, for both *npm* and PyPI packages, we will rely on available open source vulnerability databases like *npm* Security advisories[8].

We also plan to improve the visualization part of the *ConPan* CLI by providing a variety of charts to show the distribution of the number of vulnerabilities, bugs and missed updates for each installed package.

```
tracked_packages.head(3)
```

	name	package	version	major_lag	minor_lag	patch_lag	latest	outdate
0	node:stretch	unique-slug	2.0.0	0	0	1	2.0.1	1
1	node:stretch	clone	1.0.4	1	1	2	2.1.2	4
2	node:stretch	through2	2.0.3	1	0	3	3.0.1	4

Fig. 3. A Pandas dataframe showing the name and version of three *npm* packages installed in the *node:stretch* Docker image at the date of 5 March 2019, and how outdated they are (in terms of missed major, minor and patch versions).

Appendix

Release 1.0.0 of *ConPan* is available on GitHub[9] under the GPL license. For now, when the data is not available online (e.g., in the case of Debian Archive), we provide the needed data within the tool repository after extracting it using an ad-hoc script. Thus, after finishing the extension of the tool to third party packages, we will implement the ad-hoc script that we use to extract the historical data of Debian packages into the tool.

ConPan relies on third-party Python libraries such as *requests* and *psycopg2* to connect to the PostgreSQL *Ultimate Debian Database*. It also requires Docker to be installed. To facilitate its use, we plan to release *ConPan* as a Docker image, thereby avoiding the need to install libraries or to modify data links.

Acknowledgment

This work was partially supported by the Excellence of Science Project SECO-Assist (O015718F, FWO - Vlaanderen and F.R.S.-FNRS) and by the Spanish Government (TIN2014-59400-R, SobreVision).

[6]https://docs.npmjs.com/cli/ls.html
[7]http://registry.npmjs.org/
[8]https://www.npmjs.com/advisories
[9]https://github.com/neglectos/ConPan.

REFERENCES

[1] J. Turnbull, *The Docker Book: Containerization is the new virtualization*. James Turnbull, 2014.

[2] Docker Inc, "Docker - build, ship, and run any app, anywhere," https://www.docker.com/, accessed: 01/11/2018.

[3] Red Hat, "Red hat global customer tech outlook 2019: Automation, cloud, and security lead funding priorities," https://www.redhat.com/en/blog/red-hat-global-customer-tech-outlook-2019-automation-cloud-security-lead-funding-priorities, accessed: 25/01/2019.

[4] R. Shu, X. Gu, and W. Enck, "A study of security vulnerabilities on Docker Hub," in *7th ACM Conf on Data and Application Security and Privacy*. ACM, 2017, pp. 269–280.

[5] J. Gummaraju, T. Desikan, and Y. Turner, "Over 30% of official images in docker hub contain high priority security vulnerabilities," https://banyanops.com/blog/analyzing-docker-hub/, 2015.

[6] Anchore. (2019, jan) Anchore.io. [Online]. Available: https://anchore.com/

[7] E. Grande. (2019, jan) Dagda. [Online]. Available: https://github.com/eliasgranderubio/dagda

[8] OWASP. (2019, jan) Owasp. [Online]. Available: https://www.owasp.org

[9] M. Wojcik, D. Proulx, J. Baker, and R. Roberge, "Introduction to oval," *The MITRE Corporation*, 2005.

[10] ExploitDB. (2019, jan) Offensive security exploit database. [Online]. Available: https://www.exploit-db.com/

[11] Snyk.io, "Synk.io," https://snyk.io/features/container-vulnerability-management/, March 2019, accessed: 10/03/2019.

[12] K. Crowston, H. Annabi, and J. Howison, "Defining open source software project success," *ICIS 2003 Proceedings*, p. 28, 2003.

[13] J. Cox, E. Bouwers, M. van Eekelen, and J. Visser, "Measuring dependency freshness in software systems," in *Int'l Conf. Software Engineering*. IEEE Press, 2015, pp. 109–118.

[14] T. Bui, "Analysis of docker security," *arXiv preprint arXiv:1501.02967*, 2015.

[15] A. Martin, S. Raponi, T. Combe, and R. Di Pietro, "Docker ecosystem–vulnerability analysis," *Computer Communications*, vol. 122, pp. 30–43, 2018.

[16] T. Xu and D. Marinov, "Mining container image repositories for software configuration and beyond," in *Proceedings of the 40th International Conference on Software Engineering: New Ideas and Emerging Results*, 2018, pp. 49–52.

[17] A. Zerouali, T. Mens, G. Robles, and J. Gonzalez-Barahona, "On the relation between outdated docker containers, severity vulnerabilities and bugs," *arXiv preprint arXiv:1811.12874*, 2018.

[18] A. Zerouali, V. Cosentino, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, "On the impact of outdated and vulnerable javascript packages in docker images," in *International Conference on Software Analysis, Evolution and Reengineering*, 2019.

[19] W. McKinney *et al.*, "Data structures for statistical computing in python," in *Proceedings of the 9th Python in Science Conference*, vol. 445. Austin, TX, 2010, pp. 51–56.

[20] Debian, "snapshot.debian.org," https://snapshot.debian.org/, accessed: 25/01/2019.

[21] Debian, "Security bug tracker," https://security-tracker.debian.org/tracker/, accessed: 25/01/2019.

[22] L. Nussbaum and S. Zacchiroli, "The ultimate Debian database: Consolidating bazaar metadata for quality assurance and data mining," in *Working Conf. Mining Software Repositories (MSR)*, 2010, pp. 52–61.

[23] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing in science & engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[24] Michael Waskom, "seaborn: statistical data visualization," https://seaborn.pydata.org, accessed: 25/01/2019.

[25] C. Gormley and Z. Tong, *Elasticsearch: The Definitive Guide: A Distributed Real-Time Search and Analytics Engine*. " O'Reilly Media, Inc.", 2015.

[26] T. Kluyver, B. Ragan-Kelley, F. Pérez, B. E. Granger, M. Bussonnier, J. Frederic, K. Kelley, J. B. Hamrick, J. Grout, S. Corlay *et al.*, "Jupyter notebooks-a publishing format for reproducible computational workflows." in *ELPUB*, 2016, pp. 87–90.

[27] A. Zerouali, T. Mens, G. Robles, and J. M. Gonzalez-Barahona, "On the relation between outdated package containers, security vulnerabilities, and bugs," in *International Conference on Software Analysis, Evolution and Reengineering*, 2019.