OGMA: Visualisation for Software Container Security Analysis and Automated Remediation

Alan Mills
Computer Science Research Centre
University of the West of England
Bristol, UK
alan.mills@uwe.ac.uk

Jonathan White
Computer Science Research Centre
University of the West of England
Bristol, UK
jonathan6.white@uwe.ac.uk

Phil Legg
Computer Science Research Centre
University of the West of England
Bristol, UK
phil.legg@uwe.ac.uk

***Abstract*—The use of software containerisation has rapidly increased in academia and industry which has lead to the production of several container security scanning tools for assessing the security posture and threat of a container image. The variability between tools often differ on the coverage of vulnerabilities, their assessed severity and their output formats. It is also common to find duplicate Common Vulnerabilities and Exposures (CVEs) in their reporting which can often skew the risk assessment of a container. These issues along with the lack of automated solutions for maintaining up-to-date patching of container images are currently open issues identified by the research community that we address in this paper. We present *OGMA*, a visualisation tool for improved analysis and assessment of container security issues across multiple, often conflicting, scanning tools. In addition to severity, our approach helps to examine attack vector and exploit availability, while also removing duplicated CVEs, therefore providing a clearer picture for risk analysts to understand the threat posed by container deployment. Furthermore, we couple this with a novel remediation scheme for updating vulnerable containers whilst ensuring that functionality is preserved, and show how our visualisation system can highlight the improved security posture of the fixed container. Our results highlight the existing security issues in pre-built container images and the inconsistencies between scanning tools, whilst our proposed approach helps to identify and mitigate such threats to improve container security as part of the wider challenges of software supply chain security.**

***Index Terms*—Containerisation, Virtualisation, Software Vulnerabilities**

I. Introduction

Docker [1] has gained widespread adoption for containerisation, having initially been released in 2013. Specifically, Docker Hub is a widely used platform, described as 'the world's largest library and community for container images', with over 100,000 container images [2]. Whilst having readily-available containers can offer great convenience for software development teams, there is the wider issue of software supply chain security that has increased in recent years, with notable cases such as Solarwinds Orion that resulted in 18,000 of their customer organisations being compromised through their software platform. Therefore, how do we ensure that the integration of pre-built containers can be managed in a safe and secure manner? How do we ensure that we are not unknowingly introducing additional attack vectors into our operational environment, so that we can assess and manage the associated risk of pre-built containers?

Various scanning tools have been proposed to examine the security risks associated with a container to identify known Common Vulnerabilities and Exposures (CVEs), including Docker Scan [3], Dagda [4], Clair [5], Grype [6], Sysdig [7], and Trivy [8]. However, our research and experimentation with these tools shows that there is significant variation in terms of the output and coverage reported when comparing multiple scanners against the same set of containers (e.g., what CVEs are identified and the associated severity). The output of the scanning tools is often a large, text-based format that is time consuming to examine, and so there is scope for more intuitive presentation of this information to aid human analysis. Furthermore, outputs often contain many duplicated CVEs (e.g., if the same CVE is registered against several binaries or the same binary multiple times) which can result in an inflated number of CVEs and a false representation of the true security threat. Finally, it is often difficult to assess the "real-world threat" based on the standard output of these tools, and the manner in which the containerised application will be deployed within the software supply chain. For example, a critical CVE that requires local access may in reality pose less of a threat than a medium-level remote access CVE.

In this paper, we introduce a visualisation approach for improved assessment of containerisation security vulnerabilities across multiple scanner tools. Our approach, that we refer to as *OGMA*, provides a unified and holistic view of container security that can aid human analysis and highlights key factors such as attack severity, attack vector and exploit availability (as informed by Exploit-DB [9]). Our approach enables analysis on a per-scanner basis, or in a unified approach, as a means to help inform decision-making around the introduction of a container to a software stack. This allows an analyst or developer to quickly and easily disseminate the containers vulnerabilities, reducing the current burden required to create useful reporting from existing scanner solutions. Furthermore, we also introduce an automated remediation tool that we refer to as *BORVO*, that can automatically update container images where fixed binaries are available. Our initial experimentation found that this can reduce the number of CVEs by as much as 65% without impacting usability.

II. Related Works

A 2016 study by Shu *et al.* [10] presented DIVA (Docker Image Vulnerability Analysis) framework which they used to analysis over 350,000 container images. The framework utilised the Clair scanner, and found on average 180 vulnerabilities across the official and community images tested, with over 80% of these images having at least 1 high severity vulnerability. They also noted an update delay of 200 days for 50% of their test images (increasing to 400 days for 30% of images). In a later study by Liu *et al.* [11] they found similar delays in container patch management, with an average delay of 422 days for patches to be applied to images. These studies highlight that the requirement for automated patch management has been long standing and continues to be an ongoing problem in container security.

In [12] the authors expanded on an earlier thesis where they analysed 2500 docker images from official, verified, certified and community images. They compared the vulnerability findings for images within each category. Whilst official images were deemed to be secure, still they found 45.9% of images to have one or more critical or high severity CVE, and only 17.8% containing no CVEs at all.

While analysis from [10]–[12] highlights the prevalence of CVEs in docker images and the issues delayed patch management causes, they do not utilise a range of scanners, often relying on a single scanning tool such as Clair or Anchore. It is therefore quite possible that the output results presented in these studies would have been different had multiple scanner tools been used (and their combined outputs unified), as shown in our evaluation.

Tundr-Onadele *et al.* compared static and dynamic analysis, comparing Clair to multiple machine learning algorithms [13]. They found that Clair could only detect 3 out of 28 tested vulnerabilities, compared to 22 out of 28 for dynamic analysis. However, the combination of static and dynamic analysis produced the best results with 24 out of 28. In [14] the authors analysed 59 container images using three open source container scanners (Clair, Anchore and Microscanner). Their results highlight the differences in coverage and accuracy between the tools analysed and advocates a combination of static and dynamic analysis. While both studies evaluate the performance of different scanning techniques, they do not provide a comparative approach that our approach offers in terms of the scanner capabilities, coverage and output.

The authors of [15] present a framework for use in Cloud environments and deployed CI/CD (Continuous Integration / Continuous Deployment) pipelines that combines static and dynamic analysis which includes a threshold for CVE count and score. They make use of both Clair and Anchore to identify and categories vulnerabilities. Their system aims to identify vulnerabilities, and restrict usage where a user-defined vulnerability scoring threshold is met. However, this is still limited in terms of analytical investigation of the threat, and also does not provide any automated remediation to address the vulnerabilities.

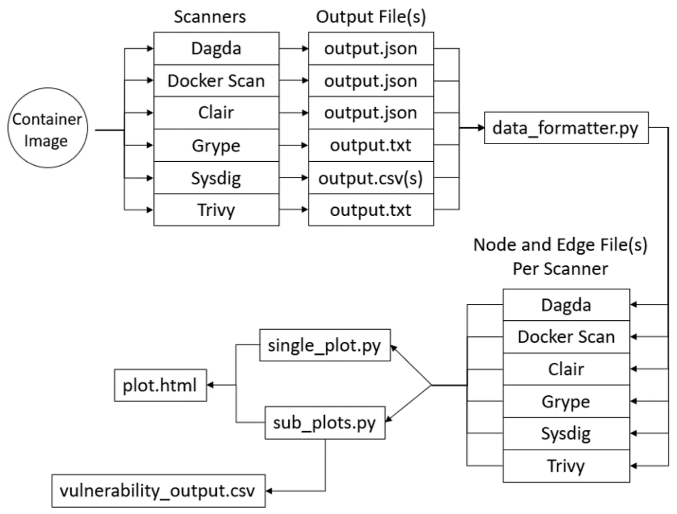

Fig. 1: OGMA workflow.

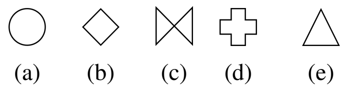

Fig. 2: Glyph design: (a) Packages (b) Local CVEs (c) Reserved CVEs (d) Remote CVEs and (e) Other Vulnerability.

In the work by Wong *et al.* [16] they take a holistic view of the threats and open security concerning containers and container deployment. One issues raised within this work is that "To date, there are many container image scanning tools [...] but few research into their effectiveness, their gaps and their impacts to the container's security". They also look at the issues surrounding container patch management and identify the current risk posed by delayed software updates and that this creates an open security issues which needs to be addressed, "[...] rapid patching is important to address vulnerability in the container before an attacker gets into it [...] Therefore, a reliable and rapid patching framework for containerized application is a gap which should be tackled quickly" [16]. We believe that our work is able to address this need in a scalable manner as highlighted in earlier research.

III. Unified Visualisation of Container Security

For our experimentation, we analyse a variety of readily-available containers from Docker Hub using six of the most common scanning tools currently available: Docker Scan [3], Dagda [4], Clair [5], Grype [6], Sysdig [7], and Trivy [8]. We wanted to first assess the variability in existing scanning results, and secondly we wanted to explore how visualisation can improve the presentation of multiple scanning results to better assess both the initial vulnerabilities that may be present within a container, and afterwards once automated remediation has been performed. We focus our analysis on official images within Docker Hub, since these tend to be packages that are widely utilised to integrate within existing software stacks and therefore receive a high number of downloads. Previous

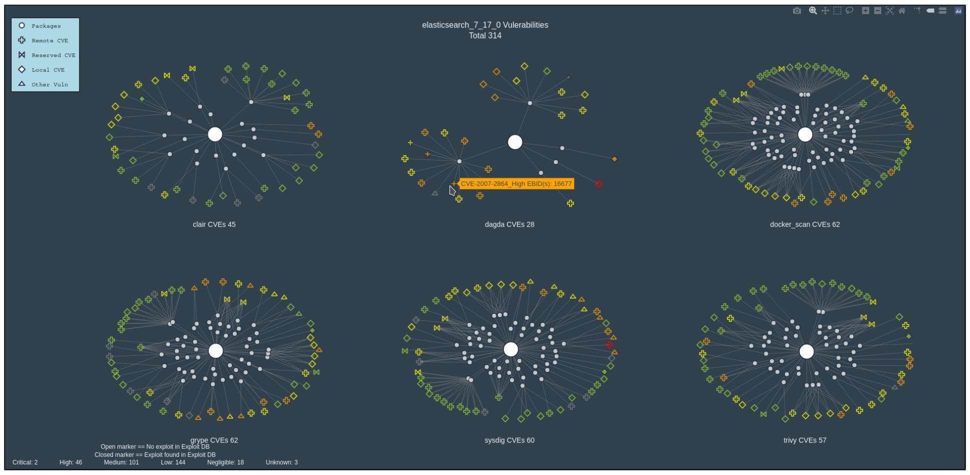

Fig. 3: OGMA sub plot visualisation for comparison of scanner performance. (Elasticsearch:7.17.0). Top: Clair, Dagda, Docker Scan. Bottom: Grype, Trivy, Sysdig.

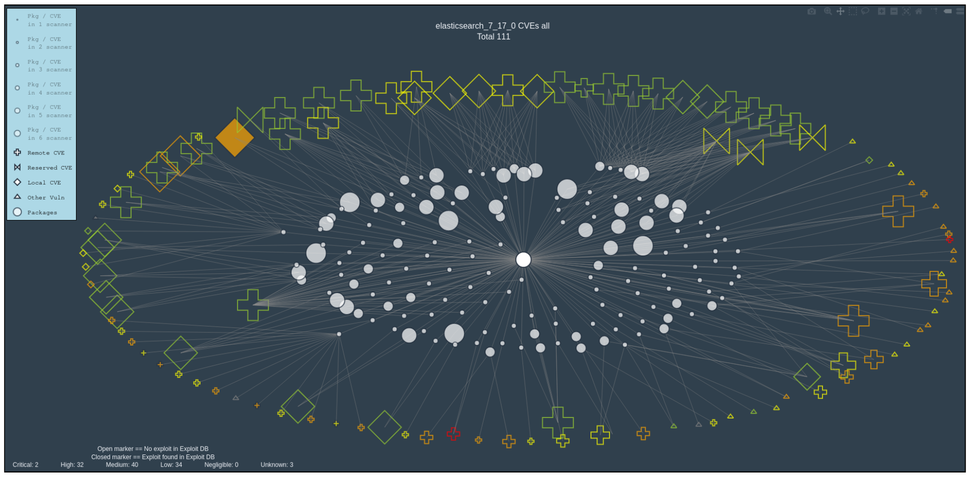

Fig. 4: OGMA single plot visualisation for unified risk assessment of identified vulnerabilities. (Elasticsearch:7.17.0)

studies have found that official images tend to have a lower number of vulnerabilities compared to verified and community images [12], which would align with our expectation.

OGMA, referring to the mythological notion of "eloquence and communication" [17], is a visualisation module that can unify and plot the output of multiple scanning tools, providing a more intuitive means of examining original and fixed security issues relating to containers. Figure 1 shows the process, whereby a container image is first scanned by all available scanning tools, and their different output file formats are fed to a data formatting script to prepare these for visualisation. For the output file of each scanner, we extract the details related to the vulnerabilities and the impacted binaries, along with the attack severity (as defined by that specific scanner) where available. If the attack severity is not reported, we call the NIST (National Institute of Standards and Technology) API [18] so that this lookup can be automated. We generate a data schema for rendering a force-directed node-link diagram for each scanner output, which can then be utilised to render either a series of subplots, or a single unified plot that shows the number of unique vulnerabilities, the severity, the attack vector (remote, local) and whether or not an exploit is available on Exploit-DB [9]. This is then accessible from the HTML output document where the user can interact with the plot to investigate the results further. Since some existing scanners do not reliably provide information regarding attack vectors and exploit availability, our process integrates additional lookups, including CIRCL (Computer Incident Response Center Luxembourg) CVE search [19] and Exploit-DB, to further enrich the scanner output and the effectiveness of the visualisation. We use a five-point colour scale as the indicator of severity: red (critical), high (orange), medium (yellow), low (green) and unknown/negligible (grey), coupled with a glyph-based design (Figure 2) for representing the packages, and the associated attack vector of identified vulnerabilities (e.g., whether the CVE can be exploited remotely or locally). The analyst can either examine the complete set of sub plots (Figure 3) to facilitate result comparison per scanner, or the unified singular

	Critical		High		Medium		Low		Negligible		Unknown	
Scanner	Vuln	EDB	Vuln	EDB	Vuln	EDB	Vuln	EDB	Vuln	EDB	Vuln	EDB
Clair	0	0	3	0	13	0	7	0	50	1	11	0
Dagda	2	0	19	2	30	3	3	0	1	0	0	0
Docker Scan	0	1	3	0	5	0	78	0	0	0	0	0
Grype	4	0	6	0	13	0	8	0	48	1	5	0
Sysdig	0	0	14	0	24	0	7	0	176	17	13	0
Trivy	4	0	8	0	15	0	57	1	0	0	11	0

TABLE I: Tabular output of OGMA highlighting the number of vulnerabilities found per scanner and the number of these vulnerabilities that have exploits available from ExploitDB (EDB). (Example for container image: Nginx:1.21.6)

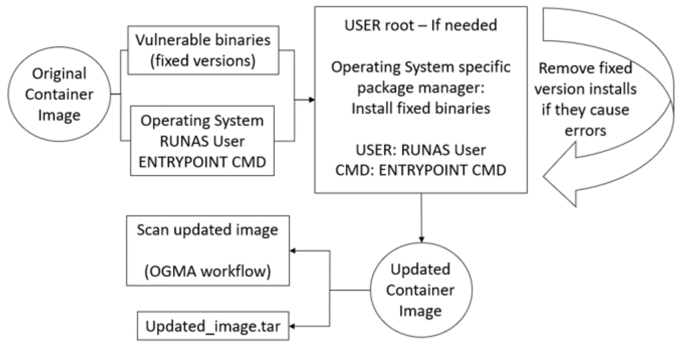
Original Container Image
Vulnerable binaries (fixed versions)
Operating System RUNAS User ENTRYPOINT CMD
USER root – If needed
Operating System specific package manager: Install fixed binaries
USER: RUNAS User
CMD: ENTRYPOINT CMD
Remove fixed version installs if they cause errors
Scan updated image
(OGMA workflow)
Updated Container Image
Updated_image.tar

Fig. 5: BORVO workflow.

plot that shows the combined set of vulnerabilities (Figure 4).

For singular plots, where two scanners reported different severity levels, the visualisation will default to the highest level reported. Glyph size is also used within the unified visualisation to show vulnerabilities that had the highest coverage across multiple scanners, helping an analyst to identify agreement (and disagreement) between the scanner results. Whilst this doesn't indicate which scanners may have omitted a result, this information can be identified using the sub plot view. Further to this, our tool also provides a CSV output that provides numerical results in a consistent and comparable format across all scanners for deeper investigation. This also includes both the existence of exploits in Exploit-DB (visually represented by a closed marker) and the severity of the associated CVE (Table I).

IV. Automated Remediation to Improve Container Security

During our investigation it became clear that many vulnerabilities in the pre-built containers have fixed updates available, as identified by scanning tools. Clair, Grype and Trivy were noted as particularly effective for identifying fixed updates, and yet as identified by previous research [16] the delay in resolving vulnerabilities in pre-built containers is high. We therefore build upon OGMA to support automated remediation of containers (referred to as *BORVO*, due to the association with "healing" [17]).

Utilising the Python Docker API, we automated the process of container inspection to obtain details such as the underlying Operating System, the default user and the entry point command. This was then combined with information about the fixed binary to create an in-memory Dockerfile. This would use the original container image as a base image, and then install the available fixed binaries. Design considerations included the ability to rollback if the fixed update caused build errors, or where the default user does not have sufficient privileges, such that the root user is required for installation. The full process is illustrated in Figure 5. The workflow is designed such that package updates can be carried out in a fully automated manner as intended without modifying the default container behaviour. The targeted installation of fixed versions avoids updating software that is not required and could introduce new vulnerabilities, whilst also minimising 'container bloat' by updating packages that are unwanted or not used, as would be the case with a generic update and upgrade process.

The output of our remediation process integrates with the OGMA subplot visualisation, facilitating before-and-after comparison between the original container and the improved container having mitigated against reported vulnerabilities (Figure 6). The updated container image is also saved as a tar file to allow for validation testing, to ensure that the expected functionality had not been compromised whilst also facilitating deployment of the updated image.

V. Experimentation and Results

We selected 25 popular images from Docker Hub offical to provide a cross-section of commonly-used applications and use cases. Each image was analysed using OGMA, and then remediated using BORVO. Table II provides the full listing of container images that we examined in this study including their version numbers, as well as the total number of vulnerabilities identified for each container image across each of the six scanning tools. We also report the total number of unique vulnerabilities in each case to highlight the issue of CVE duplication. All analysis was conducted during February 2022 and March 2022, with each container image being scanned concurrently across all six scanning tools utilised by OGMA, to ensure that any differences in reporting was due to the scanner performance rather than potential changes that could occur over time in the container image.

Each container image was analysed by BORVO to automatically remediate the identified vulnerabilities. In the cases where binaries were updated, the container was then manually analysed to ensure that the application or service worked as would be expected, based on the available documentation, such

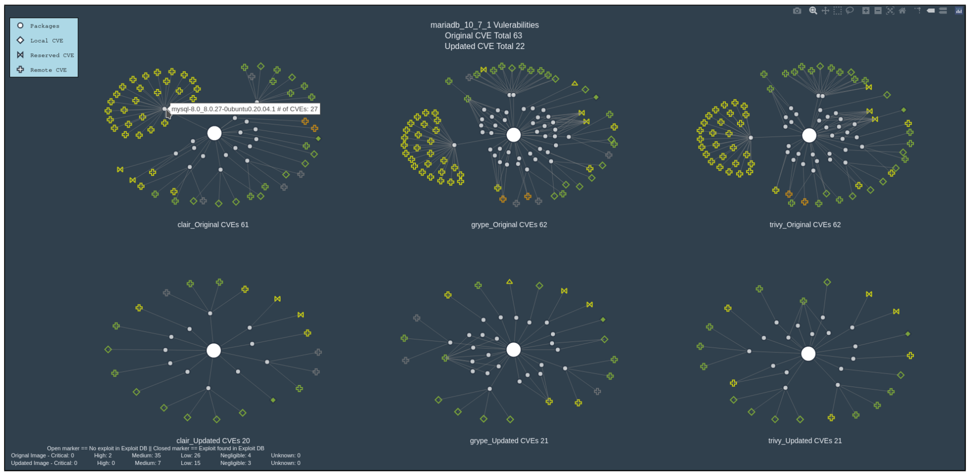
Packages
Local CVE
Reserved CVE
Remote CVE
mariadb_10_7_1 Vulerabilities
Original CVE Total 63
Updated CVE Total 22
mysql-8.0_8.0.27-0ubuntu0.20.04.1 # of CVEs: 27
clair_Original CVEs 61
grype_Original CVEs 62
trivy_Original CVEs 62
clair_Updated CVEs 20
grype_Updated CVEs 21
trivy_Updated CVEs 21
Open marker == No exploit in Exploit DB || Closed marker == Exploit found in Exploit DB
Orignal Image - Critical: 0 High: 2 Medium: 35 Low: 26 Negligible: 4 Unknown: 0
Updated Image - Critical: 0 High: 0 Medium: 7 Low: 15 Negligible: 3 Unknown: 0

Fig. 6: BORVO visualisation to highlight CVE remediation across 3 scanners (Clair, Grype and Trivy). Top row: Beforehand with 63 CVEs (2 High, 35 Medium). Bottom row: Afterwards with 22 CVEs (0 High, 7 Medium). (Mariadb:10.7.1)

as that provided by Docker Hub. It is noted that the analysis of images created using BORVO followed our initial investigation based on the scanning results from OGMA. However, we do not consider this to be an issue since our results for OGMA address the variability across different scanning tools, whilst our example shown for BORVO addresses before and after remediation on the same container image.

Table II shows the results of OGMA. Both Clair and Dagda had relatively low CVE duplicate reporting, however they also reported fewer CVEs overall in comparison to other scanners. Docker Scan had the highest level of CVE duplicate reporting, this is in part due to instances where the same CVE and same binary are reported multiple times with the only difference being the “from” section. With these duplicates instances removed Docker Scan reported the second lowest average overall for unique CVEs.

Our visualisation approach aims to improve the clarity and presentation of this information over the traditional text-based output, whilst unifying this across multiple scanners and removing duplication. Furthermore, crucial information such as attack vector and existence of an exploit is not included in the traditional scanners, yet can be easily incorporated within our visual schema. To elaborate on the conflict of reported information provided by the scanners we can examine CVE-2019-3843/CVE-2019-3844 as an example (based on Set User ID/Set Group ID vulnerabilities). Grype and Trivy both reported high, Clair and Sysdig both reported medium, Docker Scan reported low, and Dagda reported neither CVE. This difference is attributable to the source each scanner uses. NIST use the Common Vulnerability Scoring System (CVSS) that lists both as high severity, whilst Red Hat lists them as medium severity and Snyk (used by Docker Scan) low severity. Both CVEs also have verified exploits available from Exploit-DB (EDB-ID 46760), which greatly increases the realistic security threat posed by these CVEs since it lowers the barrier for entry required for an attacker to exploit.

Our BORVO results revealed that this approach was able to reduce the number of CVEs per image by approximately 27 while maintaining functionality. In some cases, such as Joomla or Vault the reduction was minimal (1 CVE) while in others the reduction was substantial, such as Mariadb where the number of CVEs was reduced by approximately 65% from 63 to 22, which also resolved all high severity CVEs (2) and 28 medium severity CVEs. In a similar instance, the number of CVEs for phpmyadmin was reduced by 174 (49%), from 353 to 179. This included 10 critical severity CVEs and 23 high severity CVEs, including three with listed exploits; CVE-2019-18276, CVE-2019-3843/CVE-2019-3844. While the results from BORVO will be dependent on the output available to it (i.e., the presence of a fixed binary to utilise) these initial results do highlight that automated container image patch management can be achieved in a manner that preserves the use case of the original image whilst mitigating against a significant number of prior vulnerabilities that may be present. This enables end users to take ownership of their own patch management in a fashion that has a low barrier to entry, without reliance on the original authors of the container image.

VI. Limitations

OGMA as a framework can be expanded to include additional reporting input, either for more scanner tools or more reporting types (such as .yaml). This was outside the scope of the current work, but would ensure longevity for OGMA.

Further and more extensive testing of BORVO, specifically the updated images it produces is required. Limited testing was conducted to ensure that immediate and obvious breaking changes had not been introduced in line with official guides and instructions (where available). However this did not cover all potential use cases and some specific or edge cases uses may have been negatively impacted. Such extensive testing was outside the scope of this paper.

	Clair		Dagda		Docker Scan		Grype		Trivy		Sysdig	
Container Image	All	Unique	All	Unique	All	Unique	All	Unique	All	Unique	All	Unique
aerospike:5.5.0.9	180	176	13	11	1409	176	410	176	410	179	709	341
centos:centos7.9.2009	31	20	382	301	783	522	691	461	775	518	993	543
chronograf:1.9.3	110	110	18	13	857	109	267	113	266	111	530	251
drupal:9.2-php7.4	194	194	64	59	970	102	381	188	380	189	962	574
elasticsearch:7.17.0	45	45	32	28	353	62	117	62	106	57	173	60
ghost:4.36.1	46	46	59	54	424	77	529	483	106	69	240	144
golang:1.16	208	208	60	55	921	97	391	202	390	203	949	563
groovy:4.0.0	31	31	36	32	243	32	102	33	66	31	108	34
httpd:2.4.52	61	61	59	54	371	61	99	61	99	62	279	188
influxdb:2.1.1	99	99	6	5	641	99	211	119	177	100	418	211
joomla:php8.0	193	193	64	59	964	101	380	187	378	187	967	579
jruby:9.3.3.0	161	161	74	66	406	65	241	170	230	160	685	482
kibana:8.0.0	53	53	17	16	331	53	133	91	105	66	0	0
logstash:8.0.0	37	37	17	16	352	44	168	102	89	37	126	63
mariadb:10.7.1	62	62	22	21	406	62	101	62	95	62	105	64
matomo:4.7.1	191	191	64	59	933	97	366	184	365	185	924	550
mongo-express:0.54.0	10	10	0	0	84	14	68	54	35	25	40	24
nginx:1.21.6	84	84	60	55	379	84	127	84	126	84	367	234
node:17.5	346	346	88	78	2686	234	937	341	914	341	2205	769
perl:5.34.0	304	304	88	78	2539	210	860	296	842	298	2136	732
phpmyadmin:5.1.3	192	192	64	59	943	99	367	185	366	186	927	552
postfixadmin:3.3.10	218	218	64	59	1022	107	408	212	406	213	949	561
python:3.9.10	349	345	89	78	2714	237	940	342	920	343	2764	831
tomcat:9.0.59	83	83	74	66	583	82	166	84	162	82	459	220
vault:1.9.4	1	1	4	4	3	1	11	11	3	3	0	0
Average	131	130	60	53	853	113	338	172	312	151	720	342

TABLE II: Analysed Images - OGMA results

VII. Conclusions and Future Work

This research highlights the challenges associated with security of software containerisation, notably that many publicly-used containers carry security vulnerabilities, and that current scanning tools often provide inconsistencies in their reporting of these vulnerabilities. We therefore present two new tools to help identify, investigate, and mitigate container security vulnerabilities, and we have demonstrated their effectiveness against existing academic and industry practice. OGMA provides unified reporting across multiple scanning tools to visualise associated attack vector, attack severity and the exploit availability across the complete set of identified vulnerabilities. BORVO provides automated container patch management and remediation. Our initial testing shows successful reduction of vulnerabilities whilst preserving container functionality. These tools help to address current, open security research questions [14] [16] by facilitating "research into [container scanner tools] effectiveness, their gaps and their impacts to the container's security" and introducing an automated solution to provide "a reliable and rapid patching framework for containerized application". Future work will involve a comprehensive study into public container vulnerabilities, and developing these initial concepts of risk-based container analysis further.

Both tools described in this paper are publicly-available under the MIT license: https://github.com/amills157/tuatha_de.

References

[1] Docker, "Home - docker," https://www.docker.com/.

[2] ——, "Docker hub - docker," https://www.docker.com/products/docker-hub/.

[3] docker, "scan-cli-plugin," https://github.com/docker/scan-cli-plugin, 2022.

[4] eliasgranderubio, "dagda," https://github.com/eliasgranderubio/dagda, 2021.

[5] quay, "clair," https://github.com/quay/clair, 2022.

[6] anchore, "grype," https://github.com/anchore/grype, 2022.

[7] sysdig, "Sysdig secure," https://sysdig.com/products/secure/, 2022.

[8] aquasecurity, "trivy," https://github.com/aquasecurity/trivy, 2022.

[9] O. Security, "Exploit database - exploits for penetration testers, researchers and ethical hackers," https://www.exploit-db.com/.

[10] 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*, 2017, pp. 269–280.

[11] P. Liu, S. Ji, L. Fu, K. Lu, X. Zhang, W.-H. Lee, T. Lu, W. Chen, and R. Beyah, "Understanding the security risks of docker hub," in *European Symposium on Research in Computer Security*. Springer, 2020, pp. 257–276.

[12] K. Wist, M. Helsem, and D. Gligoroski, "Vulnerability analysis of 2500 docker hub images," in *Advances in Security, Networks, and Internet of Things*. Springer, 2021, pp. 307–327.

[13] O. Tunde-Onadele, J. He, T. Dai, and X. Gu, "A study on container vulnerability exploit detection," in *2019 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2019, pp. 121–127.

[14] O. Javed and S. Toor, "Understanding the quality of container security vulnerability detection tools," *arXiv preprint arXiv:2101.03844*, 2021.

[15] K. Brady, S. Moon, T. Nguyen, and J. Coffman, "Docker container security in cloud computing," in *2020 10th Annual Computing and Communication Workshop and Conference (CCWC)*. IEEE, 2020, pp. 0975–0980.

[16] A. Y. Wong, E. G. Chekole, M. Ochoa, and J. Zhou, "Threat modeling and security analysis of containers: A survey," *arXiv preprint arXiv:2111.11475*, 2021.

[17] M. Lurker, *A Dictionary of Gods and Goddesses, Devils and Demons*. Routledge, 2015.

[18] NIST, "API vulnerabilities," https://nvd.nist.gov/developers/vulnerabilities.

[19] The Computer Incident Response Center Luxembourg, "CVE-search," https://www.circl.lu/services/cve-search/.