Review Paper of the State of the Art of Container Image Scanning in CI/CD pipelines

1. Executive Summary of Findings

In general, there are two different sources of container images - internal and external. Internal images are created inside the company while external ones are provided by vendors. The typical problems in internal images are outdated packages, not following best practices. External packages can also contain CVEs and execute malicious scripts.
Overall, there are two main approaches to inspect containers - static and dynamic scanning. Static scanning is useful for scanning for CVEs and checking for best practices. These issues can be quickly scanned in the pipeline. Dockerfile can contain use of multiple languages and command line tools, this makes it harder to create best practices tools.
We analysed both approaches and the future direction in using these approaches in enterprises.
The static approach is faster, but not as effective. The dynamic approach requires more resources.

2. Introduction

Containers revolutionized application delivery. Previously the companies had to validate that two applications could be installed and run together on the same virtual machine, provide detailed installation instructions for each flavor of Linux system and do testing with each cloud provider. The containers isolate applications from each other and have all the packages required for running applications. Docker containers focused on usability and provided great flexibility for the application developers. This had a toll on reducing security and lack of patterns for installing packages. The existing continuous compliance testing is not applicable for the containers since it requires an additional agent running with the application. New scanning tools were developed.
This literature review analyses the current state of the container and container images scanning.
Overall, there are three main groups of problems that require scanning: misconfiguration that leads to the increased container image size and potential security issues, outdated packages that may contain Common Vulnerabilities and Exposures(CVEs), malicious scripts and applications. These problems can be identified with two kinds of scanning - static and dynamic. Static scanning is used to scan Dockerfiles for misconfiguration and container images for known CVEs. Dynamic scanning detects malicious programmes and attacked containers.

Best practices for container images are not well-defined (Tak et al., 2018), (Henkel et al., 2020). These practices differ from the ones applied for virtual machines due to protection by container runtime and often hard to define due to overlap between multiple languages.
CVEs of different severity are present almost in any container image(Mahboob and Coffman, 2021, Brady et al., 2020), the existing tools can not always detect such issues (Tunde-Onadele et al., 2019). The security experts usually provide additional threshold or automatic analysis for CVEs found to allow the image to pass the security checks pipeline.
The static scanning is not enough, so the companies provide or create their own tools for dynamic scanning. The problem with the current state of dynamic scanning is that it allows false positives(Tunde-Onadele et al., 2019) and the attackers might hide the attack script during the virus scan (Brady et al., 2020).
A different approach could be to limit the commands that containers can run and use a more secure trusted execution environment to ruin the images (Mahboob and Coffman, 2021). This approach is best to implement in zero-trust environments.

3. Security Analysis of Container Images Using Cloud Analytics Framework

The article (Tak et al., 2018) analyses the top ten thousand most downloaded public container images from Dockerhub. The authors use a set of tools to check for vulnerabilities as well as checking the installed system for security practices compliance.
The article proposes multi-step scanning with the following tools: CRC (Compliance Rule Checker), VPC (Vulnerable Package Checker) and SNP (Security Notice Parser).
Compliance Rule Checker has four main groups of rules: SSH, Password, Permissions, extra packages and one with the remaining extra rules. SSH rules disallow installing SSH and configuring SSH inside the container. Container engines such as Docker and containerd support connecting into the containers using native mechanisms. Configuring SSH access lets the attacker by-pass authentication and increase the attack vector. The permission compliance security practices are used for hardening virtual machines to reduce attack scope. Container orchestrators usually prevent these practices from exploiting. For example, Kubernetes allows to specify that the file system will be read-only, so the rules C1, C2, C4-C23 do not make sense with such configuration. The extra binaries and processes such as telnet, rssh and FTP should not exist, except for the test purposes.
During the next step, Vulnerable Package Checker finds the packages installed in the container image and provides it as an input to the Security Notice Parser. Security Notice Parser collects package vulnerability information from various sources and verifies if the vulnerability exists in the installed packages. The system verifies only installed packages and does not validate the binaries added in the container.
The result of the scanning of most pulled images shows that only 8% of the images did not have vulnerable packages installed. The median number of vulnerable packages on the image was seven, with most vulnerable packages installed on image with Ubuntu and Debian operating system.
Compliance rule check found less than 1% of images without violations. On average there were 5.2 compliance violation per image. However, some of the violations are not strictly applicable to containers such as auto log-out and passwd configuration.
However, it must be noted that the paper does not consider packages manually installed without the package manager on the images and does not validate other files for vulnerabilities. To conclude, Compliance Rule Checker with the reduced number of rules can be useful for people who are not familiar with container images or are just learning image scanning technologies.

4. Learning from, Understanding, and Supporting DevOps Artifacts for Docker

The article(Henkel et al., 2020) focuses on extracting useful rules from the existing docker image repositories. The extracted rules reduce image size, improve build and deploy latencies. The same process can be used to extract the rules from the other custom DSL files that are used by DevOps tools such as Jenkins or Terraform. This research is one of the first ones that analyses such files.
People usually make good enough configurations that can be used in their work and do not invest in best practices. The perceived benefit in becoming an expert is not large enough since the DevOps work is mostly invisible to the rest of the company.
The researchers created a tool to extract the practices from the code and used it on Dockerfiles. There is an existing linting tool - hadolint, but due to complexity of Docker and possibility to include multiple languages the tool is missing some rules that help to reduce image size.
For the rule extraction, the researchers cloned 900000 GitHub repositories, extracted Dockerfile ASTs and minimised the number of leaves in ASTs. After that, the tool selected the leaves that execute fifty top most used command line tools. This reduces the number of leaves and simplifies the analysis. Separately, the researchers analysed Dockerfiles in docker-library repository and checked the git history to get the context for each change. This helped the authors create an initial set of rules. The docker-library GitHub organisation is owned by Docker and the changes to the repos are usually validated by the experts, so the Dockerfiles in these repos already contain the best practices more often. The result of that extraction can be formulated in 9 golden rules.
Later, the researchers ran their rule extraction tool on all repositories and compared the extracted rule with the manually collected rules. The tool was able to find 26 rules, 6 of which overlapped with the manually extracted golden rules, 4 were ungeneralizable and impossible to implement, 6 were syntactic and 4 novel semantic rules that can be used to improve dockerfiles.

5. A Study on Container Vulnerability Exploit Detection

The paper(Tunde-Onadele et al., 2019) analyses detection of vulnerabilities inside the Docker containers. There are two main approaches - ahead of time using static analysis with Clair command-line tool and the dynamic detection inside the running container by collecting system calls using Sysdig Falco tracing tool and five statistical approaches provided by it: K Nearest Neighbors(k-NN), Principal Component Analysis with k-NN, K-Means and two machine-learning based approached that Self-Organising Map (SOM).
The Clair tool inspects the containers layer-by-layer, gets the packages and their versions from the image, and then compares packages and their versions with the data in open CVE databases.
The Falco tool uses unsupervised anomaly detection algorithms that analyse the system calls.

For the experiment, the researchers took twenty-eight different existing vulnerabilities and verified images containing the vulnerabilities using the Clair tool. After that they executed an attack on the running containers to exploit CVEs and compared dynamic detection methods. As a result, Clair only detected three vulnerabilities in the images. Dynamic detection results varied from 7 to 22 for different approaches with different detects found using different approaches.
Later in the article the authors focus on 15 CVEs that cause returning a shell and executing arbitrary code since those are not detectable with a different type of monitoring, such as a container restarts counting and virtual machine metrics. Clair did not detect those CVEs, but Falco did.
Most importantly, the researchers compare the detection time using different approaches and accuracy of these detections. In particular, the SOM approach over system call frequency vectors was able to detect all attacks, but at the same time was among the slowest ones. It took about 28 seconds on average to detect each CVE. Meanwhile, the k-means approach was able to detect vulnerabilities in eleven cases, for seven of which the detection was immediate and for the rest four it took only a second to detect the attack.
However, the paper does not analyse the false positives for the regular applications. Authors mention and calculate the false positive rate for the containers with the vulnerabilities, but do not define how they calculate it. Unfortunately, the paper also does not propose the ways to reduce the false positives. It mentions that Clair can report hundreds or thousands of CVEs for each image and does not explain how to filter these issues. It does not analyse the extra used resources for each of the approaches as well.

6. A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework

This paper(Mahboob and Coffman, 2021) focused on improving container security pipeline and processes. It focuses on DevSecOps as three separate roles - development, security and operations that work together but have separate well-defined duties. Tools provide automation to reduce the human interface. The tools also scoped to a Kubernetes cluster for portability. There is an additional layer of security - Intel’s Software Guard Extension that provides a trusted execution environment (TEE). TEEs are a processor-based technology that secure code “in-process” by encrypting memory and page tables. The hardware supports secure access to the system even for the attackers with administrative privileges. TEEs are especially important in Zero Trust environments providing security in spite of CVEs. The Asylo tool is used to simplify running applications in TEEs.
The authors used Tekton as a CI/CD tool. It is a Kubenetes-native tool as its primitives are defined as Kubernetes objects. The authors split the pipeline into two separate parts. The first pipeline builds the image in Kubernetes with the Kaniko tool using Asylo to provide confidentiality of the image and push it into the container image registry. Kaniko is also a Kubernetes-native tool. It runs a simple container artifact that deploys to the cluster as an executing object. The scanning is either a part of the pipeline or provided by the registry itself.

The second part gets the newly built image from the registry and updates the Kubernetes deployment. This pipeline can be replaced with a pipeline that updates the image version in the deployment repository and the GitOps tool.
The separation of duties allows to shift the main security focus to the security team. The security team chooses tools to attest the security of the artifacts. The Trusted Execution Environment allows to shift the security concerns from the developers to the tools and thus reduce human involvement. It must be noted that image building with TEEs is significantly slower, but does not change the code execution or development time. The system is completely deployed with Kubernetes. Kubernetes decouples the pipelines from the hardware and simplifies the deployments. It also increases security and allows running the tools in so called air-gapped environments.
The paper does not analyse the possibility of deployment of their pipeline in a different cloud. The benefits of simplifying deployment of all of the components in a single cluster are not important for the real production multi-cluster environments. There are no limitations for the Asylo tool mentioned and it is not clear if the tool can be used only for compiled languages or if it can be used for scripting languages as well. It is also not clear if the Asylo tool can be deployed on existing Kubernetes clusters or if it requires specially configured nodes with custom system packages.

7. Docker Container Security in Cloud Computing

This paper(Brady et al., 2020) summarises working with containers in big organisations and proposes automated ways to analyse images for developers to prevent using malicious images.
There are two categories of files to examine and detect compromised code - static and dynamic analysis. These two types can be applied to scanning container images.
Scanning images for CVEs is a part of static analysis. It is commonly used in industry. Additionally, static analysis can include scanning source code and verifying signatures of file names, hashes and file types. There are two tools that are used for static container images scanning CoreOS Clair and Anchore Engine.
Dynamic analysis takes longer than static. In addition, it requires a container to run. The authors recommend running the container in a sandbox, so it does not impact the production services.
The researchers implemented a pipeline to validate the image and to scan it for the malware. Additionally, they implemented an internal service that allows developers to validate any image that simplifies the consumption of third-party images inside the organisation. The service triggers the pipeline and notifies the developers about the result.
The pipeline consists of four layers. On the first layer CoreOS Clair checks the image for CVEs, then the number of CVEs found is compared with the threshold. The next layer includes scanning with Anchor Engine. If the image passes these checks, it is pushed to the test registry, where the dynamic pipeline picks it up. The first step of the dynamic analysis is the virus check. There are cases when the malware evades antivirus emulators, so the system starts the container in privileged mode. To reduce the scope of attack Docker-in-Docker is used and the container is isolated. A separate container uses tcpdump to record all network communication. Docker logs the file system changes that are used to find new binaries added to the container. The validation runs for several hours and the logs

are saved into the S3 bucket, so they can be checked later in case of the false negative result.
Due to the multilayer approach the pipeline can be extended and modified if needed. For example, the new tools can gate the publishing of the image in the test registry or be added to the dynamic validation.

8. Summary of Findings

The containers change the requirements for security. The images are being constantly built, the container lifetime is short, so the automated checks are required for more efficiency and faster detection of vulnerabilities. The image scanning pipeline is used in all organisations that use containers (Mahboob and Coffman, 2021).
The existing static analysis tools are still in their early stages, the best practices are still being defined and require specialists for verifications. The image size can be reduced for the majority of images by applying such practices(Henkel et al., 2020). The image size reduction will improve the container startup and image build time.
Static security analysis is provided by several open-source tools and it is not sufficient enough since it requires either manual processing of existing CVEs or automatic threshold(Brady et al., 2020). The CVE scanning does not detect all CVEs in the container and does not protect against not detected vulnerabilities (Tunde-Onadele et al., 2019). This requires additional dynamic scanning and additional runtime protection.
There are two ways to do dynamic scanning - validate the image before deploying it to the environment (Brady et al., 2020). The container should be started in a restricted environment that can not affect the production environment and all its communication should be analysed and saved. This approach is especially beneficial for the external images and can work even against sophisticated attacks.
Another approach is to run the dynamic analysis libraries against containers in production. This way detects attacks on the existing applications and can help to protect against unknown CVEs in valid packages. This approach is not well-studied, since there can be many false positives that can prevent automated blocking of attacks and many false negatives that can allow the attack to proceed. . The extra load on the running processes is also not well defined yet.
Since it is impossible to detect all vulnerable packages, there is a need to improve security in-spite-of-CVEs and use zero trust environments. This requires hardening of applications, images and container runtimes. The work is started by the hardware vendors, but the application developers still do not use secure processing(Mahboob and Coffman, 2021). The additional frameworks that automate and simplify such protection are needed.

Literature:

1. Brady, K., Moon, S., Nguyen, T. and Coffman, J. (2020). Docker Container Security in Cloud Computing. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/abstract/document/9031195.

2. Henkel, J., Bird, C., Lahiri, S.K. and Reps, T. (2020). Learning from, understanding, and supporting DevOps artifacts for docker. Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering.
3. Mahboob, J. and Coffman, J. (2021). A Kubernetes CI/CD Pipeline with Asylo as a Trusted Execution Environment Abstraction Framework. 2021 IEEE 11th Annual Computing and Communication Workshop and Conference (CCWC).
4. Tak, B., Kim, H., Suneja, S., Isci, C. and Kudva, P. (2018). Security Analysis of Container Images Using Cloud Analytics Framework. Web Services – ICWS 2018, 10966, pp.116–133.
5. Tunde-Onadele, O., He, J., Dai, T. and Gu, X. (2019). A Study on Container Vulnerability Exploit Detection. 2019 IEEE International Conference on Cloud Engineering (IC2E).