Docker Container Security in Cloud Computing

Kelly Brady,* Seung Moon,* Tuan Nguyen,* Joel Coffman*†

* Engineering for Professionals, Whiting School of Engineering, Johns Hopkins University
Email: { kbrady16, smoon22, tnguy185, joel.coffman }@jhu.edu

† Department of Computer and Cyber Sciences, United States Air Force Academy

***Abstract*—Docker is popular within the software development community due to the versatility, portability, and scalability of containers. However, concerns over vulnerabilities have grown as the security of applications become increasingly dependent on the security of the images that serve as the applications' building blocks. As more development processes migrate to the cloud, validating the security of images that are pulled from various repositories is paramount. In this paper, we describe a continuous integration and continuous deployment (CI/CD) system that validates the security of Docker images throughout the software development life cycle. We introduce images with vulnerabilities and measure the effectiveness of our approach at identifying the vulnerabilities. In addition, we use dynamic analysis to assess the security of Docker containers based on their behavior and show that it complements the static analyses typically used for security assessments.**

***Index Terms*—containers, vulnerability analysis, continuous integration / continuous deployment (CI/CD), Docker**

I. Introduction

Containers have gained significant traction within the software development community because they allow developers to avoid the time consuming configuration of libraries and dependencies. An image is a file that contains the required code, configuration, and libraries to run an application. Containers are instances of these images with every instance having the same underlying dependencies. Because they only encompass the instructions and code that are necessary for the application to run, containers are lightweight compared to alternative approaches such as virtual machines (VMs). Multiple containers can run on a single physical or virtual machine, making them ideal for many phases of the software development cycle.

A microservice is a software architectural paradigm that is commonly used with containers. Microservices aim to make software development easier through optimal use of various resources. In essence, each software function is provisioned in a single application component. The components then use an application programming interface (API) to interact with each other. Microservices serve as building blocks to design and develop scalable and maintainable software. This approach decreases the dependencies between major software functions and minimizes the code base with which developers interact.

Docker is a popular Platform as a Service (PaaS) for containers. Many organizations such as Google and Amazon are incorporating Docker into their software development life cycle, as Docker allows developers to rapidly design, develop, test, and deploy applications while optimizing resource usage [1]. Due to Docker's popularity, the development community has created repositories of Docker images, such as Docker Hub,[1] to increase reusability and to encourage file sharing. With the paradigm presented by containers and microservices, software development is increasingly dependent on small, reusable components that are developed independently and distributed by different organizations. This dependency, in turn, raises concerns regarding the security of the entire Docker image distribution pipeline. Architects at Docker now encourage developers and publishers to include risk analysis that considers the entire distribution pipeline itself as actively malicious [1]. Introducing a multi-layered security mechanism at the Docker registry level may mitigate vulnerabilities from being introduced into Docker repositories.

The contributions of our work are two-fold:

- We implement a multi-stage continuous integration and continuous deployment (CI/CD) pipeline to evaluate the security of Docker images. Our pipeline prevents the publishing and consequent reuse of images with known vulnerabilities.
- We demonstrate dynamic analysis for for developers to use when selecting Docker images. Such dynamic analysis may be incorporated into our CI/CD pipeline, and we evaluate it using images with malicious content to show how it detects abnormal runtime behavior.

In a nutshell, our work is designed to improve security practices when using containers as part of software development.

The remainder of this paper is organized as follows. In Section II, we present an overview of various Docker mechanisms used to enforce the security of images. In Section III, we explain how our CI/CD pipeline is an extension of single-layered security tools that reduce the surface area where vulnerabilities may be introduced into the image distribution pipeline. Section IV describes the implementation of our prototype using Amazon Web Services (AWS). Section V covers our experimental evaluation. Section VI summarizes related work, and we conclude in Section VII.

[1] https://hub.docker.com/

978-1-7281-3783-4/20/$31.00 ©2020 IEEE

II. Background

Malware analysis usually takes the form of examining files or executables to detect compromises. There are two categories of this analysis—static analysis and dynamic (or behavioral) analysis. This section describes these two types of analysis as well as how they pertain to Docker images.

Historically, software and hardware vendors used various scoring metrics to measure software vulnerabilities. The resulting lack of uniformity eventually led to the creation of the Common Vulnerabilities and Exposures (CVE) system [2]. CVEs provide a framework to quantify and assess vulnerabilities and exposures, and it also enables publicly sharing such information. One common way to prevent vulnerabilities from being introduced to the distribution pipeline is to regularly scan Docker images against CVEs. Detecting vulnerabilities within Docker images encourages actions to address them [3].

Scanning for CVEs can actually be considered as a part of static analysis, but the term static analysis covers a broader set of actions. In static analysis, the content of data is examined without executing the instructions that are captured in the data. Static analysis has the capability to detect bugs in source code such as unreachable code, variable misuse, uncalled functions, improper memory usage, and boundary value violations. Static analysis also uses signatures based on file names, hashes, and file types to indicate if a file is malicious.

In comparison, dynamic analysis observes a container's behavior. Some methods of dynamic analysis are port scans before or after execution, process monitoring, recording changes in firewall rules, registry changes, and network activity monitoring. While dynamic analysis typically takes longer than static analysis, the results may be more intuitive. However, Docker containers must be launched in a confined *sandbox* so that other services and resources in production are not impacted by the container.

Although there have been some efforts across the Docker community to encourage security analysis by users, they are often ignored. Thus, it would be ideal to incorporate security analysis tools into the development cycle of Docker images. Due to the rising concerns of vulnerabilities introduced by Docker images' vulnerabilities, there are several open source tools available that may be incorporated into such as process. CoreOS Clair[2] is one such tool that performs static analysis of image vulnerabilities. Another tool that is currently available is Anchore Engine,[3] which includes CVE-based reporting. Anchore Engine's security policy enables users to have fine-grained control over security enforcement by allowing customized security policies, helping users to achieve NIST 800-190 compliance [4].

III. Design

To address these vulnerabilities and shortcomings, we propose an automated process for developers to scan and analyze images for vulnerabilities. Combining Clair and Anchor Engine into a CI/CD pipeline lessens the burden on developers by automating security analyses. In addition, we implement a service API that scans for malware on public images, which allows developers to research an image and verify its security before incorporating it into their products. Our service minimizes setup costs and allows developers perform static and dynamic analyses quicker, easier, and with minimal resource investment.

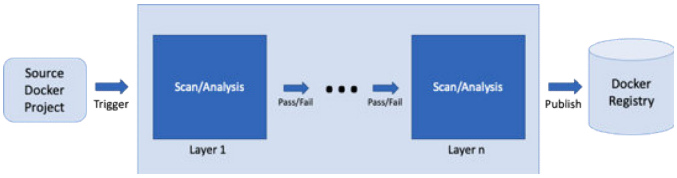

Fig. 1. CI/CD workflow for Docker image security

A. CI/CD Pipeline

We propose a multi-layered security approach for CI/CD during the development of Docker images. Our design enables developers to flexibly incorporate desired security analysis tools into their development processes. In addition, it encourages the definition of well-defined security policies for Docker images used in their development environments.

Figure 1 illustrates the workflow of our CI/CD pipeline for Docker images. The source Docker project initiates the CI/CD process. Each pipeline may contain an arbitrary number of stages with vulnerability scans and static or dynamic analysis—whatever is appropriate for the development environment. When the source code of a new Docker project enters the pipeline and on all future updates to the project's source code, it triggers the specified security checks. In order to give the developers and administrators flexibility to extend our pipeline, each layer may enforce its security checks in an arbitrary manner. If, and only if, the source Docker project passes the security checks at each layer, then the project is published to the Docker registry.

B. Dynamic Analysis

Due to the increased risk of implementing dynamic analysis on the same system as the pipeline, we propose a second tool to accomplish this goal. We implement an API to automate and capture useful information about specified Docker images to identify bugs and malicious content. This API service serves two purposes: first, it is a research tool for developers to analyze public images, and second, it serves as a second layer of our multi-layered security mechanism. Our service allows developers to analyze any public image for vulnerabilities and malware, using Clair for vulnerability analysis and scanning for malicious content using VirusTotal.[4] Lastly, it executes basic dynamic analysis by running the image for a period of time to capture file changes, network traffic, and list processes throughout the image's execution.

Malicious images often include bash scripts that create secure shell (SSH) tunnels back to a command-and-control

[2]https://coreos.com/clair/
[3]https://anchore.com/
[4]https://www.virustotal.com/

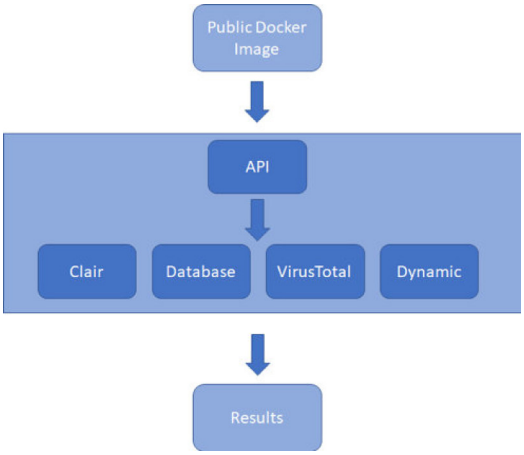

Fig. 2. API service flow for developers to research public images

server when the container runs. These malicious images try to download additional binaries and install shell code. However, only dynamic analysis captures this behavior. To address this, we developed a sub-service within the tool to automate the dynamic examination of Docker images. The process uses Docker-in-Docker (dind) to automate and capture such behavior in a sandbox. Though Docker-in-Docker does provide some isolation, it is still executed in a privileged mode and still has access to the host and has similar permissions to Docker running on the host. Thus, it is important to run this Docker-in-Docker sandbox in a hypervisor or a cloud service that provides the appropriate level of isolation.

IV. Implementation

In this section, we describe the realization of our design using AWS.

A. CI/CD Pipeline

A two-stage pipeline was implemented to demonstrate our CI/CD pipeline. Clair and Anchore Engine provide the security analyses. Figure 3 illustrates our architecture. A Virtual Private Cloud (VPC) logically isolates the various resources within the AWS ecosystem. We further isolate Clair and Anchore Engine using two availability zones. Each availability zone has a public subnet and a private subnet, and within each public subnet, the gateway allows components within the private subnet to access the Internet. A PostgreSQL database cluster spans the two private subnets.

To store the source content of Docker project, we use an AWS CodeCommit repository. Whenever the state of the repository changes, the event triggers the execution of the security layers in our CI/CD pipeline.

We perform vulnerability scanning within our first layer using CoreOS Clair. Clair provides seamless integration with our pipeline by providing API endpoints where client applications can run vulnerability scanning. Clair categorizes CVEs into seven categories: unknown, negligible, low, medium, high, critical, and defcon1 with unknown being least severe and defcon1 being most severe. Two configuration values, category threshold and vulnerability count, determine if an image passes or fails the CVE scan. If the scan finds vulnerabilities that are more severe than the threshold category and if the vulnerability count exceeds the threshold count, then the CI/CD stage fails on the image. One AWS availability zone runs services related to Clair with a Fargate Elastic Container Service (ECS) cluster in the private subnet. A load balancer distributes requests across the ECS cluster.

If the source Docker project passes the first layer's security analysis, it goes through similar steps in the second layer using Anchore Engine. Although a custom security policy my be defined, our prototype uses Anchore's default security policy, which performs light vulnerability checks (e.g., the image does not contain packages with critical vulnerabilities) and Dockerfile checks (e.g., port 22 should not be exposed). Although Anchore Engine also performs CVE-based reporting, we felt it would be insufficient to perform the entire CI/CD processes only using CVE-based reporting. Due to the microservice nature of how Anchore is deployed, a Fargate ECS cluster turned out to be infeasible. Thus, we used an ECS cluster that is capable of launching c4.xlarge Elastic Compute Cloud (EC2) instances and housing Anchore Engine's microservices.

If an image passes all the security layers, the image is published to a dedicated AWS Elastic Container Registry (ECR) repository where it is available to others who have access to that repository. This operation is performed within the final layer of our pipeline.

Our implementation significantly improves upon prior work [3], [5]. First, multiple layers are incorporated into our CI/CD pipeline. Second, it requires significantly less manual setup: for example, developers are not required to create an ECR repository for each security analysis [3]. Using our implementation, developers and administrators are able to provision all of the required services and resources for their CI/CD process using a single AWS CloudFormation template. In addition, our load balancers use auto-scaling so that the system scales horizontally based on load.

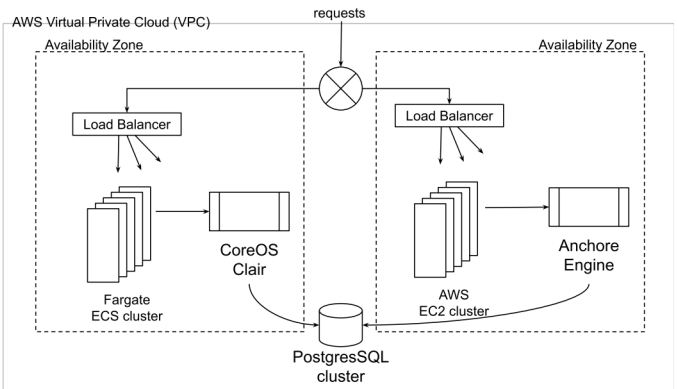

Fig. 3. AWS architecture diagram for backend services

B. Dynamic Analysis

A simple way to determine if an image contains malicious code is using a virus scanner. We use VirusTotal because of the large number of virus scanners that it supports. The downside of relying on antivirus is that most virus scanners only perform static analysis and look for signatures while some do perform dynamic analysis with their own emulators. However, there is evidence that malware is starting to evade antivirus emulators [6]. By running the image dynamically in Docker-in-Docker, the image is running within a privileged mode so there is no emulation. The image is essentially running as if it is on the host, which should allow more accurate analysis.

In order to capture traffic of an image, another container with tcpdump runs alongside the target container. These two containers share network settings. Starting the tcpdump container requires a few seconds before it captures traffic, so any initial traffic may be lost. There are some cases where the container of interest does not run long enough for tcpdump container to start. In cases of malicious images, the containers often run for longer periods of time because they need to communicate back to a control server for further instructions. We save the network traffic in a packet capture (pcap) file and push it to Simple Storage Service (S3) for later analysis.

Docker has the option to log file system changes, which we also use in our dynamic analysis. The difference is the changes made to the base image from when the container was running. This log helps when analyzing malicious images that download and execute new binaries that are not part of the original image. Another analysis capability is listing all the processes in the container. The dynamic portion of the API performs a process listing every second during the run.

V. Evaluation

This section evaluates our design, starting with the CI/CD pipeline before progressing to the dynamic analysis.

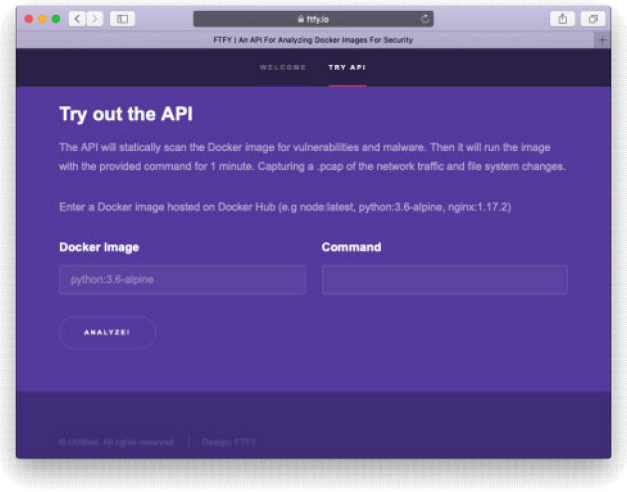

Fig. 4. Screenshot of ftfy.io, the web portal for dynamic analysis

TABLE I
Docker images used in the evaluation

Image	Tag
postgres	11.5
ubuntu	18.04
python	3.6-alpine
node	10.16.0-alpine
nginx	1.17.2
buamod/eicar	latest
zoolu2/jauto	latest

A. CI/CD Pipeline

We use two Docker projects to evaluate our CI/CD security pipeline (see Table I). One project, an nginx website[5] that exposes port 80 and 443 for HTTP and HTTPS traffic, is the same as used in prior work [3]. We chose PostgreSQL 11 as our second Docker project because we believed that the image would be relatively secure against vulnerabilities. For CoreOS Clair, an image passes if, and only if, the image did not flag fifty or more vulnerabilities that were categorized as "high."

Figure 5 displays the vulnerability count that was produced by CoreOS Clair in our CI/CD pipeline. The result validates our prediction of PostgreSQL 11's security: the image was not flagged for any vulnerabilities. The nginx website image, however, was flagged for 143 total vulnerabilities. Even with all these vulnerabilities, the image still passed because it did not exceed the threshold of fifty or more vulnerabilities that were "high" or more severe.

The second layer of our pipeline enforces Anchore Engine's default security policy, which includes light vulnerability checks and Dockerfile checks. The PostgreSQL 11 image met all of the security requirements of the default security policy and was published to our Docker registry. Conversely, the sample nginx website did not pass the security policy and was not published to our registry.

[5]https://github.com/aws-samples/aws-codepipeline-docker-vulnerability-scan

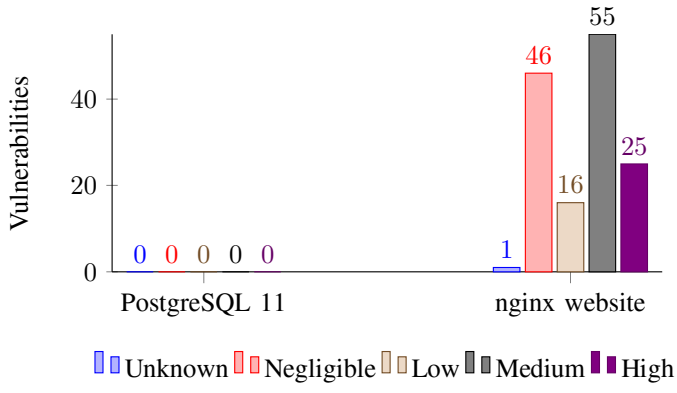

Fig. 5. Clair vulnerability scan results (lower is better).

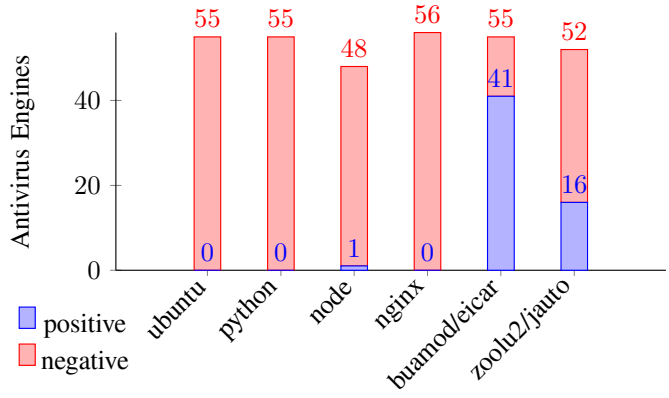

Fig. 6. VirusTotal scan results. A positive scan indicates that an antivirus indicated the image contained malicious content.

B. Dynamic Analysis

Clair did not detect any vulnerabilities for the images listed in Table I, which was expected because the images were the most up to date images available. Vulnerabilities are not expected unless the developer adds an outdated package to the base image. Additionally the dynamic analysis is not focused on finding vulnerabilities.

The VirusTotal results, summarized in Table 6, were also not surprising. The ubuntu, python, node, and nginx images are officially supported by Docker Hub and therefore have most likely been analyzed for malware prior to being published. We are unsure why node was flagged by one antivirus vendor, but we believe it is a false positive. The last two images, buamod/eicar and zoolu2/jauto, were expected to be positive because of prior known malicious content. The buamod/eicar image contains a file with a binary signature that is used to test virus scanners, and the zoolu2/jauto image contains scripts that download and install a cryptocurrency miner.

When we examined the images for testing using dynamic analysis within the API tool, results were not interesting for the ubuntu, python, and node images. Further discussion focuses on the nginx and zoolu2/jauto images. Figure 7 summarizes the

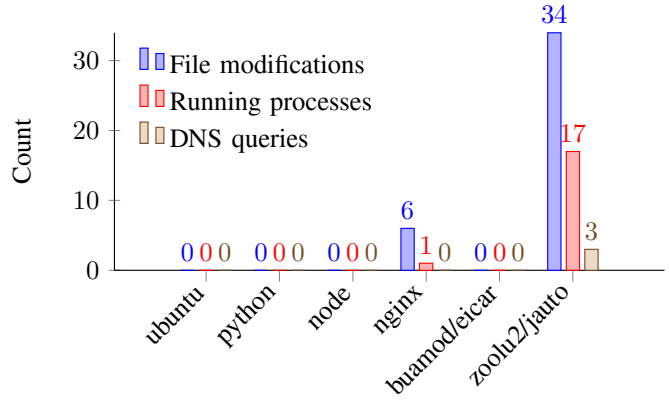

Fig. 7. Summary of results using the dynamic analysis API

number of file modifications, running processes, and Domain Name System (DNS) queries for each image following 30 seconds of execution.

The file system changes relate to the functionality of the image and what actions the image is trying to perform. It is especially helpful when examining malicious files and in the case of the test images. For example, the zoolu2/jauto image writes two scripts, `cmd.sh` and `cmd1.sh`, in `/root`. The image also surveys the system and saves information such as CPU metadata. This image also performs some IP searches and scans using Shodan, a search engine for Internet connected devices. A larger red flag appeared when the image started SSH and Tor daemons. In comparison, nginx's file changes comparise temporary files, all of which reside in `/var/cache/nginx`.

Furthermore, the dynamic API scanner also uploads a pcap of the network analysis while the image runs. Many of the images did not have any network traffic and consequently has empty pcap files. From the DNS records that are acquired in this analysis, we can tell that the zoolu2/jauto container authenticates with shodan.io to perform IP address searches to find open ports and services running on the host. The next notable finding is a DNS request for us-east.cryptonight-hub.miningpoolhub.com, a place for cryptocurrency miners allocate mining resources. Although we did not fully analyze all the network traffic, the evidence suggests that this image is malicious and is attempting to mine cryptocurrency. Our analysis suggests that this image should not be trusted for execution outside of a sandbox.

C. Limitations

Our current scheme relies heavily on vulnerability scan count. Even though this decision serves as a good initial phase for DevSecOps processes, an arbitrary number of vulnerabilities is insufficient for production-level security requirements. Additionally, Anchore's default security policy used during our experiment may not define all of the security requirements that must be met by an organization.

The API service for dynamic analysis currently supports the scanning of public images but does not support uploading of Dockerfiles for analysis. Custom images are usually designed by the developer and therefore the content of the Dockerfile should be known. However, there are cases where official libraries and dependencies can be tampered with, which jeopardizes the overall security of the image. We propose to expand the API service to include the scanning of custom images. In these cases the dynamic analysis portion of the API service can help monitor and detect such cases. From our results it is evident that safe Docker images should pass at least a virus scan and make few modifications to the file system.

VI. Related Work

Many tools exist to scan for CVEs. For example, OpenSCAP[6] checks for vulnerabilities based on information from the National Vulnerability Database[7] and violations of organizational security policies. oscap-docker is a tool for scanning Docker

[6]https://www.open-scap.org/
[7]https://nvd.nist.gov/

images. We could easily incorporate such tools in our CI/CD pipeline, but similar functionality is provided by CoreOS Clair and Anchore Engine, and our CI/CD pipeline is extensible, not being limited to only CVE scanning. The Docker Trusted Registry offers Docker Security Scanning, which scans images for vulnerabilities listed in a CVE database. Regrettably, this service requires an enterprise license and additional security scanning extension. Our CI/CD pipeline could be deployed by organizations who desire a free alternative to this service.

Adethyaa and Jernigan [3] demonstrate a CI/CD process for Docker images that uses AWS resources. Valance [5] performs similar analysis using the Anchore Engine. Both use a single stage security mechanism (CoreOS Clair or the Anchore Engine respectively) that executes static security analysis on Docker images. A major limitation of Adethyaa and Jernigan's work is the requirement for manual provisioning of the AWS services and an inability to define custom security policies. Valance's approach lacks a source that initiates the entire CI/CD process and lacks auto-scaling for the static analyses. In comparison, our approach is extensible, supporting complex workflows with multiple security analysis tools, and is scalable.

Related to our dynamic analysis, Wan et al. [7] sandbox containers by mining rules based on legitimate system calls enountered in automated testing; in production, the sandbox restricts system calls that have not been whitelisted. CIMPLIFIER [8] uses dynamic analysis to debloat (i.e., remove unnecessary files from) containers and partition them according to the principle of least privilege. Both sandbox mining and CIMPLIFIER require automated tests to identify required resources, and, in later work [9], static analysis and symbolic execution improve coverage when the automated tests are incomplete. The dynamic analysis used in these works is comparable to our own (i.e., recording system calls and files being accessed), but our dynamic analysis tool is designed for users to explore an unknown container where automated tests may not be available. Thus, our goal differs in that our focus is the exploration of a container rather than hardening one.

VII. CONCLUSION

Developers using containers are currently vulnerable to malware and lack tools that effectively quantify this risk. Existing tools, while effective, are time consuming and challenging to implement and may introduce new risks if not implemented correctly. Our work addresses these issues by creating user-friendly tools to detect vulnerabilities and malicious code.

Our results show that virus scans and dynamic analysis are effective at detecting malicious behavior in Docker containers. In particular, safe images show few file modifications, running processes, and DNS queries whereas malicious images tend to download and execute files not initially present in the image. By using our API service for dynamic analysis, developers are better equipped to make decisions regarding which base images to use. Automating the static and runtime checks also frees developers to build more secure applications.

A. Recommendations

There is currently no formal process to report malicious images to Docker Hub. It would be beneficial for Docker Hub to create a reporting process and to investigate such images. In previously reported cases, Docker Hub took over eight months to act and remove an account, potentially allowing users to pull malicious images unknowingly in the interim [10]. Creating a dedicated reporting mechanism not only mitigates the risk of such attacks but also encourages users to be wary of what they are downloading and possibly to perform their own security checks.

B. Future Work

We would like to integrate our dynamic analysis with our CI/CD pipeline, which requires creating a sandbox for Docker images so that it cannot affect other services or resources. Exploring how to use Anchore Engine to perform dynamic analysis within our CI/CD process may be worthwhile. Another interesting extension is to apply machine learning to the dynamic analysis portion of the API service. Instead of users manually reviewing file system changes and network traffic logs that the API outputs, machine learning automates the classification of images as malicious or benign based on their activity. The file modifications, running processes, and DNS queries are an ideal starting point for signatures.

REFERENCES

[1] S. Winkel, "Security Assurance of Docker Containers: Part 1," *ISSA Journal*, April 2017.

[2] P. Mell, K. Scarfone, and S. Romanosky, "The Common Vulnerability Scoring System (CVSS) and Its Applicability to Federal Agency Systems," National Institute of Standards and Technology, Tech. Rep. Interagency Report 7435, August 2007.

[3] V. Adethyaa and T. Jernigan, "Scanning Docker Images for Vulnerabilities using Clair, Amazon ECS, ECR, and AWS CodePipeline," AWS Compute Blog, November 2018, online: https://aws.amazon.com/blogs/compute/scanning-docker-images-for-vulnerabilities-using-clair-amazon-ecs-ecr-aws-codepipeline/.

[4] J. Valance, "Using Anchore Policies to Help Achieve the CIS Docker Benchmark," Anchore Blog, May 2019, online: https://anchore.com/cis-docker-benchmark/.

[5] ——, "Adding Container Security and Compliance Scanning to your AWS CodeBuild pipeline," Anchore Blog, February 2019, online: https://anchore.com/adding-container-security-and-compliance-scanning-to-your-aws-codebuild-pipeline/.

[6] J. Blackthorne, A. Bulazel, A. Fasano, P. Biernat, and B. Yener, "AVLeak: Fingerprinting Antivirus Emulators through Black-Box Testing," in *10th USENIX Workshop on Offensive Technologies*. Austin, TX: USENIX Association, Aug. 2016. [Online]. Available: https://www.usenix.org/conference/woot16/workshop-program/presentation/blackthorne

[7] Z. Wan, D. L. Lo, X. Xia, L. Cai, and S. Li, "Mining Sandboxes for Linux Containers," in *Proceedings of the 2017 IEEE International Conference on Software Testing, Verification and Validation*, ser. ICST '17, March 2017, pp. 92–102.

[8] V. Rastogi, D. Davidson, L. De Carli, S. Jha, and P. McDaniel, "Cimplifier: Automatically Debloating Containers," in *Proceedings of the 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: ACM, September 2017, pp. 476–486.

[9] 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*, ser. FEAST '17. New York, NY, USA: ACM, November 2017, pp. 51–56.

[10] D. Goodin, "Backdoored images downloaded 5 million times finally removed from Docker Hub," Online: https://arstechnica.com/information-technology/2018/06/backdoored-images-downloaded-5-million-times-finally-removed-from-docker-hub/, June 2018.