Applying Threat Intelligence to Containers

Author: Brandon Helms, bhelms85@gmail.com
Advisor: Tanya Baccam

Abstract

The products and services businesses build, and support continue to evolve, and so does the infrastructure used to build these technologies. Containers have become a trendy way for developers to run their applications. This trend is primarily due to the ease of use, repeatability for deploying, and the resource reduction cost that happens when running a container-based environment. The unfortunate side effects of this tend to be seen on the security side, where developers pick container images based on keyword searches or "most-used" and do not have a straightforward way to see the risks the container image exposes based on the current threat environment. Although security tooling continues to advance around container security, the community is still creating a false sense of security by using outdated practices on their containers with little focus on the attacker's perspective.

1. Introduction

Society has advanced to needing the internet for services and products. However, to help facilitate the delivery of new applications, developers rely heavily on infrastructure that will provide them with capabilities to iterate at a high velocity. To do this, they require standardization, repeatability, and infrastructure to support their application's needs.

Over the years, a single application run on a dedicated physical server, to being able to virtualize the operating system so that an application can run on a single virtual machine and multiple virtual machines can run on a single physical server. While this was a considerable improvement in minimizing the physical resources needed for an organization, virtual machines still required many resources to run an application. Society has now advanced its infrastructure capabilities to support the concept of "containers." Containers are a lightweight, standalone, executable package of software that includes everything needed to run an application while attempting to minimize the resources needed to support the application. This advancement has allowed engineers to build applications that run on minimal physical resources, which reduces organization costs.

1.1. Why Containers are the New Normal

Containers do not only provide the ability to reduce the costs of physical resources; they also provide a new way for developers to interact, debug, and iterate over their applications. Unfortunately, many teams have not invested in these security improvements and are still using legacy developer practices when interacting with their containers. It is these practices that are still being exploited by attackers today. “92% of companies are using containers in production, a 300% increase since 2016.” (CNFC Survey, 2020).

When developers build applications, they will often need external services to help make their application work. A great example would be an application that needs a proxy to route to different services within their application. In traditional physical servers and virtual machines, a developer would need to install a vanilla operating system (e.g., Windows Server/Debian/RHEL), install the proxy needed to route their traffic (e.g.,

Nginx/Caddy2), and then configure the proxy to work on the operating system and their application in a fault-tolerant way. With containers, the same developer can search across a container image repo (e.g., Docker Hub) for a container that has already been built and configured to run a proxy of their choosing, where the developer needs to update the proxy configuration to route to their application. For the developer, this saves time researching and creating this capability for their application. In addition, it speeds up the development process so that the developer can efficiently orchestrate the creation, upgrading, and configuration of these container images.

Incredible as this sounds, this process creates a new problem that most developers do not consider. The developer must implicitly trust that the creator of the container image that they are using has securely built the container image and that they did not introduce malicious software (e.g., backdoors). Furthermore, the developer must trust that the container creator will maintain the container over time. Unfortunately, these security problems typically are not calculated in risk planning when using container technology.

The security community is aware of this problem. It has invested a lot into giving developers the tools needed to assess the basic security of their containers. However, these tools primarily focus on the software vulnerabilities detected through basic vulnerability scanners such as Snyk, CLAIR, and StackRox to determine the security risk of a container and do not focus on the security risks from the perspective of an attacker or threat.

1.2. Thesis Statement

Although containers are helping developers build applications in a more standardized way while allowing them to iterate at a higher velocity than legacy solutions, containers are creating a false sense of security for organizations. This false sense of security is primarily derived by outdated developer practices and lack of diverse security tooling for containers.

1.3. Benefits and Metrics of Containers

Containers can benefit an organization by increasing iteration velocity and reducing the security risk applications and infrastructure exposed. Unfortunately, the

security risk exposure tends to be overlooked in favor of developers selecting their container images based on technical requirements with little to no focus on providing a low-security risk to the organization. Furthermore, developers tend to be the owners of the security of their applications but lack the expertise or observability to build container images using a secure configuration. Plenty of statistics show the continual security risks of not providing secure configurations for containers. Some of the key metrics include:

- 81% of developers believe developers should own security, but they need to be better equipped. (Tate, 2021)
- For every $1 cybercriminals generate through a crypto mining attack against a cloud container environment, victims end up paying a $53 bill. (Vizard, 2022)
- 61% of all images pulled come from public repositories. Over 250,000 images were scanned during the analysis, and the results showed that threat actors were actively using Docker Hub to spread malware. (Toulas, 2022)
- 85% of the container images running in production environments contain at least one vulnerability. Three-quarters of those vulnerabilities (75%) were rated as "high" or "critical." (Staff, 2022)
- A search using the port-scanning service Shodan reveals that some 6,000 IP addresses may have vulnerable installations of Docker publicly assessable. (Lemos, 2020)
- Hackers managed to exploit a vulnerability in Jenkins container to crypto mine about $3.5 million, or 10,800 Monero, in 18 months. (Ms. Smith, 2018)
- It was discovered that six malicious images hosted on Docker Hub had been collectively pulled over 2 million times. That is 2 million users potentially mining Monero. (Ashutosh, C. and Rajewar. R, 2022)

To change the common developer culture, engineers must be presented with a holistic security risk assessment that enables them to decide about using a container in a low-friction way which is currently missing from current security tooling for container images.

Brandon Helms, bhelms85@gmail.com

This security assessment will include the known software vulnerabilities detected from static scanning and standard attack patterns used by attackers that the container is susceptible to. Unfortunately, while a lot of this technology exists in different command line interface (CLI) based applications, there has not been a one-stop solution that enables a minimal developer amount of effort to see these risks as they apply to common attack frameworks (e.g., MITRE ATT&CK).

2. Research on Container Security

The phrase "security best practices" represents a highly opinionated view and distracts from how a business should be set up. Keeping this in mind, the goal for this research is to collect multiple opinions around "security best practices" when dealing with container security and consolidate them into a subset of the most discussed practices. This will hopefully remove most of the biases vendors may introduce and give a concrete set of goals against which to assess a container. It should be noted that this research focuses on the container and the images that make up the container. This research does not focus on the orchestration platform used by organizations to decide to deploy and manage their containers.

While there are over 100,000 images hosted on Docker Hub, testing has been condensed down to a subset of container images focusing on one of the most used applications in a corporate network, the web proxy. This document will refer to these selected container images as "samples." More information on the samples can be found in Appendix B.

To help facilitate the testing, Container360 was created, which focuses on conducting automated threat scans based on the MITRE ATT&CK framework for containers. While this tool is still in its infant stages, it could detect weaknesses based on the above framework and enable the exploration of container information in bulk for investigating container behaviors at runtime. Throughout the rest of this document, Container360 will be used to assist in analyzing containers and presenting a centralized and reproducible way for investigating threats against containers.

Brandon Helms, bhelms85@gmail.com

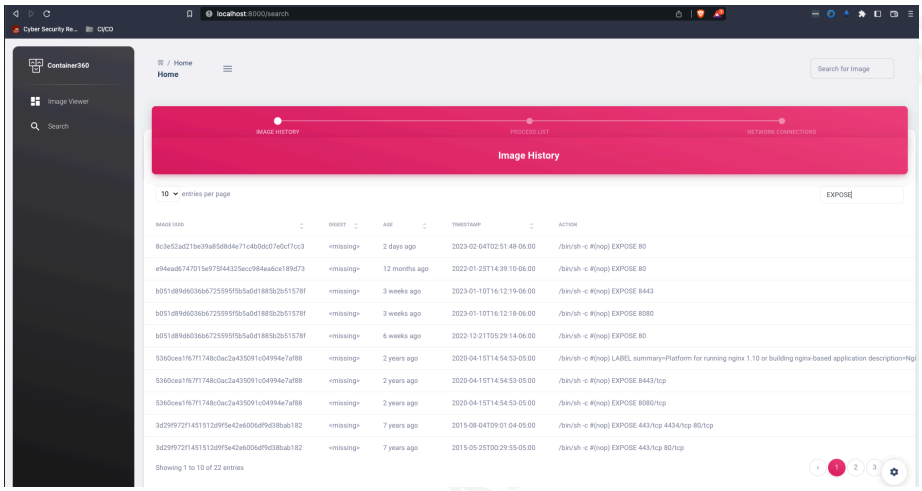

Figure 1: Container360 Bulk Image Search

The subsequent research will focus on known public attacks against organizations running containers. Unfortunately, this research will be much harder to collect data on as most organizations do not report breaches, and those that do rarely release the technical details of the attack and tend to keep their public report high level. This means that the data derived will need to be completed and account for an extremely low sample set.

Lastly, the research breaks apart the different components of a container and how an attacker can exploit the container. This includes containers created with malicious intent (e.g., Trojan backdoor) and containers created with a non-malicious intent but have known security problems backed in (e.g., static credentials for publicly exposed services). This information will be tied to the top-used images. This research should produce a complete picture of the current culture of containers and the security risks exposed to this culture while at the same time producing quantitative results that can be replicated by any future researcher wanting to validate or expand on this topic.

2.1. Introduction to Container360

Container360 takes a different approach to container security from the typical vulnerability scanners used to assess containers. For the initial proof of concept, the threat engine was mapped to align with the MITRE ATT&CK framework for containers. This information can be used to create a more comprehensive picture for the security of a container should it be deployed in a production environment outside of the traditional software vulnerability approach.

Brandon Helms, bhelms85@gmail.com

Users can run Container360 in a local environment, testing out any container image stored by Docker Hub. Once the user starts the scan for the image, Container360 will download the container image and use multiple techniques to conduct both static and dynamic analysis looking for threat identifiers that map back to the MITRE ATT&CK framework for containers. These techniques focus solely on the container image itself and not the orchestration platform that loads the container image (e.g., Kubernetes, Docker Swarm).

This tool is under active development and is subject to changes. A complete listing of capabilities, installation, and usage can be found in Appendix A.

2.2. Key Components of Container

To understand where the security risks lie within a container, engineers must understand what makes up a container and how that differs from previous technical stacks (e.g., Virtual Machines). Virtual machines and legacy servers rely on an operating system to be fully installed. After the operating system is installed, these systems will need ephemeral applications to be installed that support the developer's application. Finally, the application will be installed. Each of these steps requires a significant number of resources to be created. Orchestrating all these steps has proven difficult for developers.

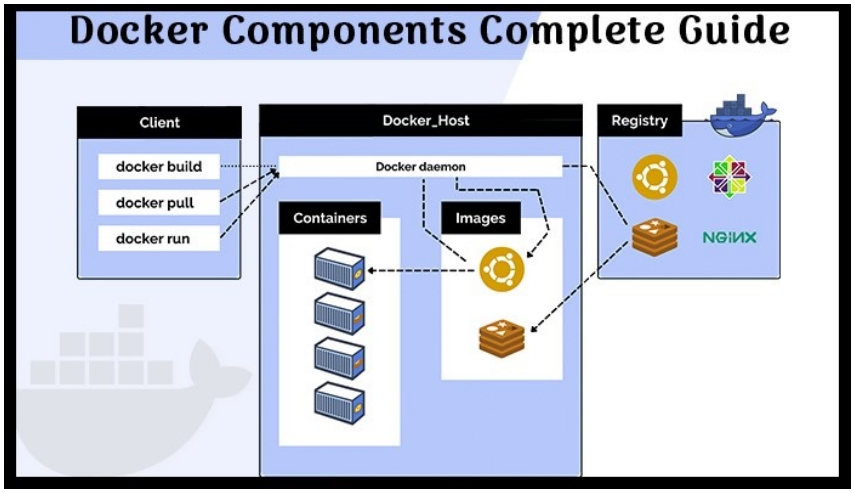

Figure 2. Docker Component Breakdown

Containers were designed to be lightweight packages of the developer's application code and dependencies, such as specific versions of programming language runtimes and libraries required to run the company's software services. By design, containers make it easy to share resources (e.g., CPU/Memory) at the operating systems level and offer a logical packaging mechanism in which applications can be abstracted from the environment they run on. This makes containers extremely portable and deployable in a repeatable process.

2.2.1. Container Registry

Container registries are centralized repositories that contain Docker images for use by a consumer. Typically, a registry will house a repository for Docker images. Any user can typically create a repository and upload their images to them.

2.2.2. Container Repository

Container repositories are a collection of images hosted by a team, organization, or user. Each image will use tags to represent differentials in their images. In addition, they may document release notes and other documentation that supports how to use and configure their images.

It should be noted that it is prevalent to see multiple repositories hosting similarly named images. This could be for both malicious and non-malicious use.

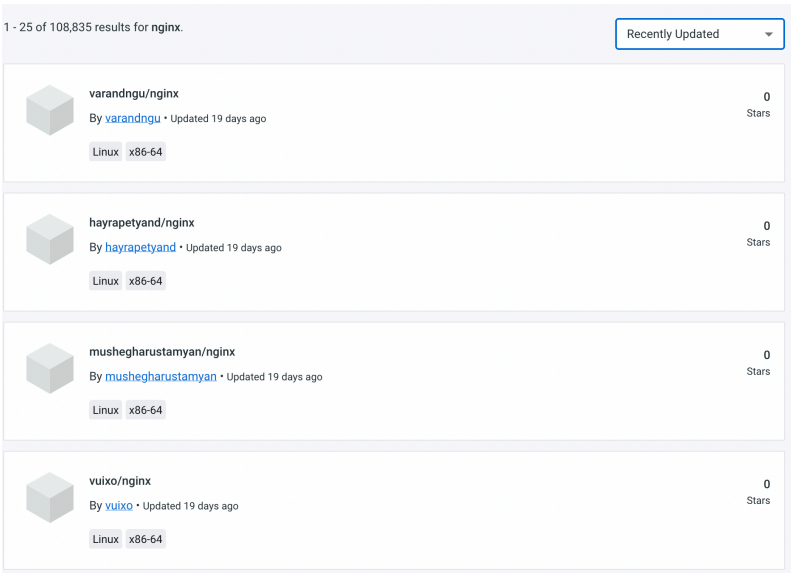

Figure 3. Images with the Same Name but Different Repository

2.2.3. Docker Image

Docker images are used to build containers and hold the metadata that elaborates the capabilities of the container using YAML. In addition, these images are used to house the resources needed to run an application.

2.2.4. Container Image Layers

One of the core aspects of the container images is that they are based on a layered filesystem. Every image comes with numerous layers, depending on the layer below it. The first layer is called the base layer, which contains the base operating system and image. Each time Docker executes a new line in the Dockerfile, it creates a new layer with the result of executing that line. It then adds that layer to the Socker image and keeps track of all the individual layers as a cache.

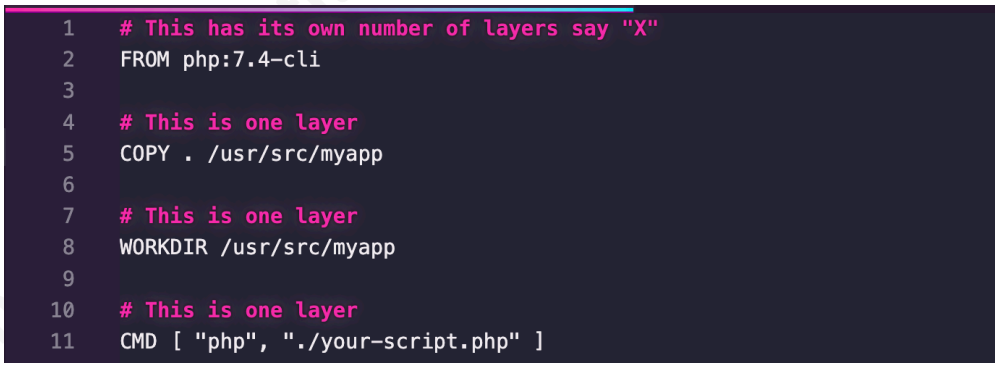

Figure 4. Dockerfile layers

Layers are more complicated because an image has multiple layers, and a newly created image can be referenced as the source for a new image. This creates layers and layers on top of an image, eventually making it hard to understand the build composure of an image. For example, if "Figure 2" was built into a new image called "new_php" with the tag "1.0", that image could be used to build a new image such as the one in "Figure 3."

```
# This has its own number of layers say "X"
FROM new_php:1.0

# This is one layer
RUN apt-get install busybox -y

# This is on layer
RUN apt-get install netcat -y

# This is one layer
CMD [ "nc", "-lvp", "443" ]
```

Figure 5. Dockerfile new_php source

3. Findings and Discussion

At a high level, the research has shown that while the use of containers is increasing, the investment in securely configured containers could be much higher.

Key findings using Container360 include:

- 43% of samples were unable to run using default settings
- 75% of samples containers run as “root”
- 50% of samples have package managers installed
- 75% of samples have tools for downloading files remotely
- 45% of samples have a scheduler tool installed
- 85% of samples can create files as the current user
- 95% of samples have at least one tool for network testing installed

Container360 attempts to run a sample using the “docker run” command with no parameters passed in. While there are legitimate reasons a container image would be built to only run when certain parameters are supplied (e.g., environmental variables), there is no programmatic way via the Docker Hub API to determine the minimal requirements to have a container run without terminating. Since these images were meant to run a web server, they should never shut down. Container360 assumes that the container image is not able to be run when it terminates.

Brandon Helms, bhelms85@gmail.com

For container images that did run long enough for threat scanning to complete, it was found that most container images run with the highest user level by default. This was determined by Container360 conducting a “whoami” on initial connection and the result returned being “root.” This does not mean that the other 25% were not privileged users or a user that did not have root-like permissions. Another major finding Container360 was able to detect showed that half the container images scanned contained a built-in package manager (e.g., apt). Container360 was able to detect this by running a `which` command looking for different package managers on the system. Failure to find a package manager does not mean that one is not installed, but that Container360 was unable to find one in the system default paths. Combining these two findings (root as default user and package manager installed) together, should an attacker land on the running container, they would have complete control and could install any tooling they needed for post-exploitation.

While the findings provide strong support that majority of container images that are used contain multiple security gaps and the repositories housing these container images do not present this information to the engineer which creates a false sense of security for the engineer and leads to untracked security issues being introduced into the business. During the research, it was discovered that there are non-related security tools that could be repurposed to help developers make more informed decisions when dealing with container security; however, these tools are manual and still require tuning and manual review by a human. As such, introducing a capability that could be integrated into these platforms could significantly improve the decision-making for selecting the best solution while minimizing the security risk to the business.

3.1. Standardizing on Security Best Practices

Defining “security best practices” around containers has become highly opinionated and results varied based on several variables, primarily around vendor blogs where they wanted to promote their solution. The most common “best practices” for container security are:

- Include as little as possible in a container image
- Use trustworthy container images and container registries
- Use immutable deployments and anchor to specific versions

- Avoid storing hard-coded sensitive data in container images
- Use telemetry and observability
- Scan container images for vulnerabilities and misconfigurations
- Apply the principle of least privilege
- Use short-lived containers to reduce the attack surface

These best practices are derived from the following sources and determined not by the publication's author but by the number of blogs that referenced the best practice.

- Anchore DevOps (2022). Conference and Blog
- Faun Publication
- Kroll Blog
- Palo Alto at Ignite Conference 2019
- SDX Central Blog
- Snyk Blog
- StackRox Blog and Video
- SysDig Blog
- ThreatStack Blog
- Tigera Blog
- Trend Micro Blog
- VMWare Blog

It should be noted that the value of each "best practice" is relative to the environment it is applied to, and not all best practices reduce the security risk to the same degree.

3.1.1. Include as Little as Possible into the Container Image

When adding to an image, teams should only add the relevant parts of their application that are needed to make the application work efficiently. This includes only the application code (compiled where possible) and dependencies needed to run that code

inside the container. Where possible, avoid using volume mounts, as this introduces a bridge from host to container.

In the samples, it was found that 50% of the container images had a package manager installed (apt/apt-get), and 75% of the samples had a tool that could download remote software using tools such as "curl" and "wget."With either of the tools mentioned above, a hacker can repurpose the tooling to download their malicious applications for use on the compromised container. It was also discovered that 85% of the container images enabled the default user to create and run files on the system. Combining the ability to create and execute files with the ability to download files remotely gives an attacker the ability to execute their malicious applications without requiring sophistication.

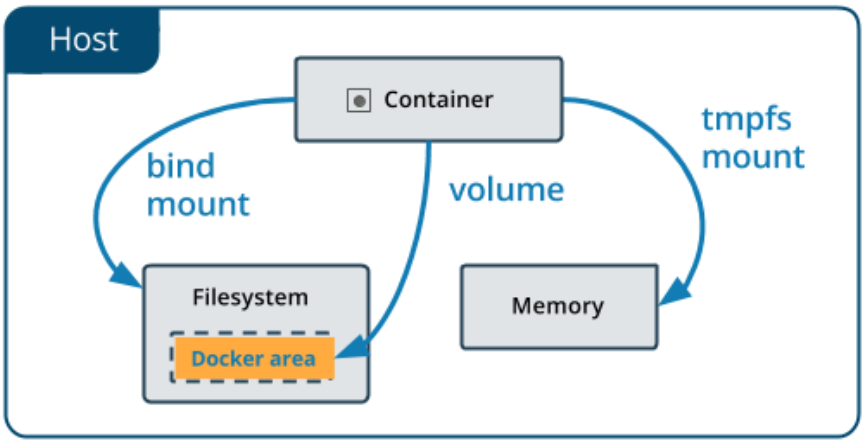

Figure 6. Volume Mounts for Containers

The best practice to prevent this is to create a new Dockerfile, reference a base image that does not contain package managers, and download tools (e.g., Google Distroless Container Images). This ensures that a team's container image will contain no added software that could enable attackers to further their attack should they compromise a company's application.

```
# Distroless Image from Google for Python3
FROM gcr.io/distroless/python3-debian11:debug

# Copying All application files to Image
COPY . /app

# Setting my working directory to /app
WORKDIR /app

# Running my application "hello.py"
CMD ["hello.py", "/etc"]
```

Figure 6. Using Distroless with Dockerfile

After a team has selected their base image, they can add their files by using the "COPY" command to bring over your application and all required files. Finally, teams can use the "RUN" or "CMD" commands to execute their application at runtime.

```
# We will use the php container image tagged to '7.4-cli'
FROM php:7.4-cli

# Copy files from your current directory to the /usr/src/myapp directory of the container
COPY . /usr/src/myapp

# Changes the directory for where we are working out of
WORKDIR /usr/src/myapp

# Execute our code
CMD [ "php", "./your-script.php" ]
```

Figure 7. Sample Dockerfile for Copying Code

Once the team has created a new Dockerfile, they will need to create a new container image and push it to a trusted registry for distribution. This practice is effective at minimizing security risks to containers.

Furthermore, having everything installed into the container image prior to running creates the possibility that the image can remove the native shell, as it should not be needed once the container image is running. A significant security risk that can break repeatability for deployments is when images install added software at runtime. This typically takes the form of package managers for either the operating system (e.g., apt) or

at a software level (e.g., pip for Python). When possible, use immutable methods for executing the application.

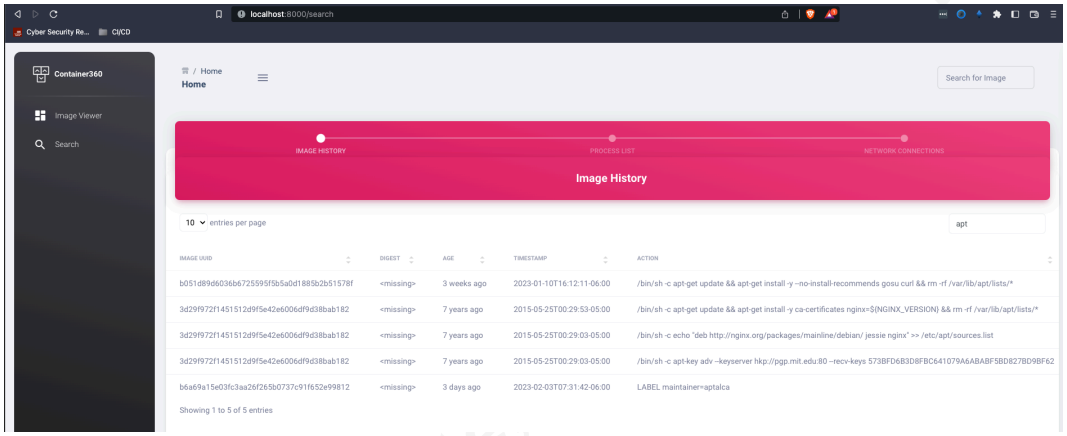

Figure 7: Package Manager Detection via Container360

From an attacker's perspective, removing the native shell from a system significantly impacts post-exploitation for an attacker as it forces them to build the capabilities and find ways to upload them to the system. Therefore, this significantly reduces risk in the "Execution," "Lateral Movement," "Discovery," "Command and Control," and "Exfiltration" techniques of the MITRE ATT&CK Framework. (*MITRE ATT&CK®*)

3.1.2. Use Trustworthy Container Images and Registries

Trustworthy is a very opinionated term, which could lead to a false sense of security. Variables that can help instill trust include:

- Trusted by the registry provided
- Verified publisher
- Official container image by the application company
- Top downloaded repositories as determined by Docker Hub
- Container images used by peers

The idea behind “trust” is that these teams have applied strong security controls in their images. As a result, these images have a low risk of intentional malicious activity and

rely on the community to find security bugs and report back to the container maintainer for patching.

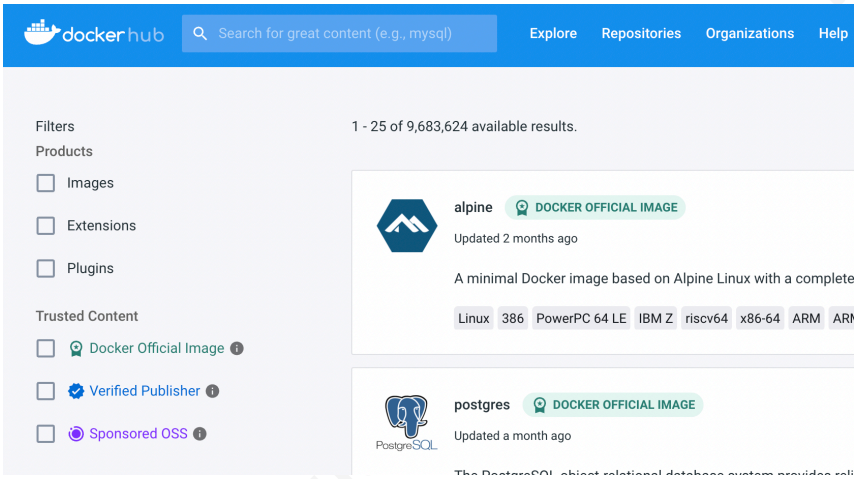

Figure 8. Docker Hub Trusted Content

Regarding container registries, the engineer should focus on registries that are not typically known to harbor malicious content. For example, the most used container repository used by public consumption is Docker Hub. According to Sysdig, over 250,000 images on Docker Hub were scanned, and the results showed that threat actors were actively using Docker Hub to spread malware.

During the selection process for the sample container images, the "Trusted Content" did not imply strong security controls. Security testing on the different levels of "Trusted Content" did not yield any noticeable security benefits. First, each level has images that failed to run in the default configuration except for the single Docker Official Image, which was able to run successfully. Each level had a 50% or more significant percentage of the container images running as a privileged user (e.g., root). Lastly, each level had greater than 35% test failures when mapping the container image to the MITRE ATT&CK framework for Containers. It should be noted that the "Not Trusted" Images had the lowest percentage of privileged user access and the second lowest test failures.

Trust Status	Failed to Run	# Tested	Privileged User	Test Failure
Docker Official	0	1	100%	47.37%

Verified Publisher	4	7	72%	38.35%
Sponsored OSS	6	8	88%	50.88%
Not Trusted	5	4	50%	43.42%

Table 1: Docker Hub "Trusted Content" Threat Testing Results

From a MITRE ATT&CK mapping perspective, the "Trusted Content" level does not directly correlate to the container's security.

3.1.3. Use Immutable Deployments and Anchor to Specific Versions

One of the more robust security controls that can be implement is to make an image immutable. This is where the engineer removes the "write" attribute from all files on the system to include virtual directories. This cannot be done within the Dockerfile and must be done by the orchestration system (e.g., Kubernetes, Docker Compose). However, building a container image in a way that supports "read-only" at runtime enables these orchestration platforms to make the image immutable at runtime.

Making an image immutable at runtime means that if the container image becomes compromised, an attacker would not be able to modify the container's filesystem. From a risk reduction, this mitigates all the “Persistence” techniques in the MITRE ATT&CK framework. It also eliminates many of the "Initial Access” and “Execution” techniques (e.g., Installing a backdoor on a website by file upload).

Anchoring to specific static tags allows an application to ensure that no changes are introduced at build time. Most container tags use a "semver" tagging system for static tags. It is recommended that developers stay away from dynamic tags such as "latest" as they can change without notice which could change the application's behavior and security profile.

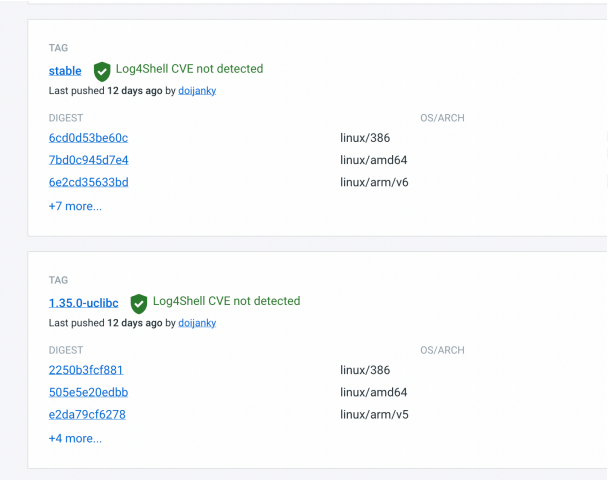

Figure 9. Docker Hub Image Tags

This is one of the strongest security measures that can be put in place to minimize the impact of an attack at the container level. Of the samples tested, 85% of the container images were configured to be immutable at the Dockerfile level, as these containers were able to have at least one (1) file manually created on them.

3.1.4. Avoid Storing Hard-Coded Sensitive Data in Container Images

Legacy-style development required developers to store sensitive data such as private keys, passwords, and API tokens on disk. Newer technologies have enabled virtual machines and containers to pull in sensitive data through API calls which can be stored as environmental variables. While these secrets can still be discovered if an attacker gains access to the container, it does mitigate the potential for discovery via static code analysis on the container image stored in a repository. This minimizes the attack surface of "Initial Access," "Credential Access," and "Privilege Access" in the MITRE ATT&CK framework. This is a vital security measure that can be put in place to minimize the impact of an attack at the container level.

None of the tested samples had evidence of hard-coded, sensitive information, including hashes stored in "/etc/shadow," credentials stored in the user's ".bash_history" or ".bashrc" files, nor passwords that could be found via OSINT.

3.1.5. Use Telemetry and Observability on Container Images

Observability is defined as a collection of logs, metrics, and traces. Typically, a container by itself will not provide observability. Instead, either the application will send observability data to a source (e.g., Humio/Splunk/ELK), or an orchestration platform will be put in place that will read the "stdout" and "stderr" of each container and send the output to a source.

Developers will want to ensure that critical metrics for both development and security are logged in a way that can be captured by observability tooling. In testing, this practice does not imply strong security.

3.1.6. Scan Container Images for Vulnerabilities and Misconfigurations

Vulnerability Scanning of software and software dependencies is one of the most invested areas for container security. However, most of these solutions focus on the static component of an image by analyzing the software installed within the container and using signatures in applications to map back to known vulnerabilities (e.g., CVEs).

This is a good starting point for reducing security risk but tends to be a high noise-generating solution. When vulnerabilities are found, it is recommended to build a new image with the fix/mitigation rather than apply a patch to the current image.

3.1.7. Apply the Principle of Least Privilege to Container Configurations

The principle of least privilege is the idea that any user, program, or process should have only the minimum privileges necessary to perform its function. Unfortunately, this can be hard to achieve and is typically a process that constantly needs reviewing.

For container security, the principle of least privilege focuses on a few areas. The main area is running the container in “privileged” mode. Privilege mode means that if a user is root in a container, they have the privileges of rooting on the host system. Some use cases require a container to be run in privileged mode, but most containers should run their applications without this privilege.

```
ENV BITNAMI_PKG_CHMOD="-R g+rwX" \
    HOME="/" \
    OS_ARCH="amd64" \
    OS_FLAVOUR="debian-9" \
    OS_NAME="linux"

# Install required system packages and dependencies
RUN install_packages libc6
RUN . ./libcomponent.sh && component_unpack "redis" "5.0.2-0" --checksum f41d595e87405f1a00b623884ae0f5c0457017cefb86cf22e7ce2a

COPY rootfs /
RUN /prepare.sh
ENV ALLOW_EMPTY_PASSWORD="no" \
    BITNAMI_APP_NAME="redis" \
    BITNAMI_IMAGE_VERSION="5.0.2-debian-9-r0" \
    PATH="/opt/bitnami/redis/bin:$PATH" \
    REDIS_DISABLE_COMMANDS="" \
    REDIS_MASTER_HOST="" \
    REDIS_MASTER_PASSWORD="" \
    REDIS_MASTER_PORT_NUMBER="6379" \
    REDIS_PASSWORD="" \
    REDIS_REPLICATION_MODE=""

EXPOSE 6379

USER 1001
ENTRYPOINT [ "/entrypoint.sh" ]
CMD [ "/run.sh" ]
```

Figure 10. Modifying Dockerfile Does Not Need Root

The sample container images had a 75% detection rate (15 out of 20) with the default user being "root" 10% detection rate (2 out of 20) where the default user was manually created (e.g., the id of 1001), and only 5% detection rate (1 out of 20) where the default user had meager privileges (e.g., nobody default account).

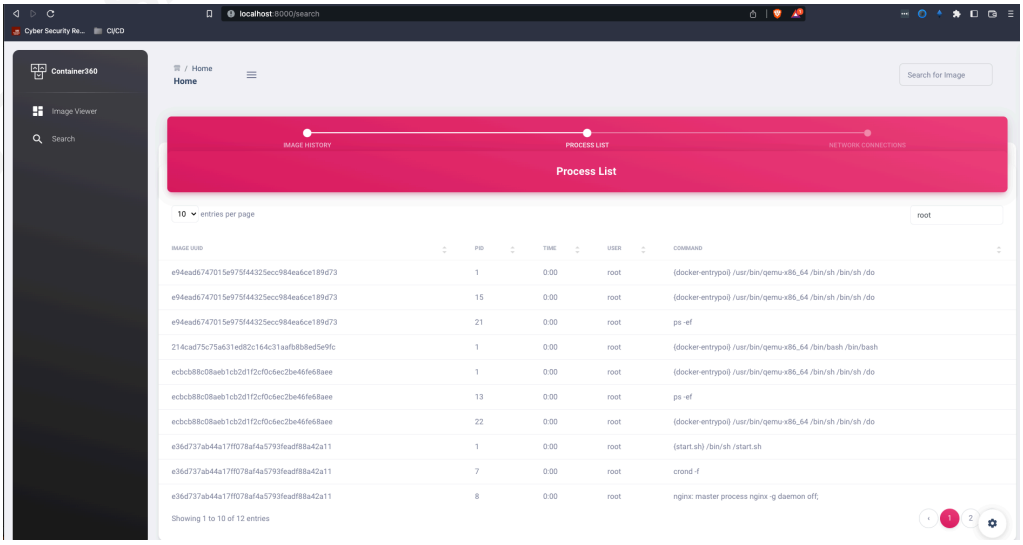

Figure 11: Root user detection via Container360

By implementing non-privileged containers and designing containers around the principle of least privilege, security risk can be reduced in the "Privilege Escalation" and "Lateral Movement" techniques of the MITRE ATT&CK Framework.

Brandon Helms, bhelms85@gmail.com

3.1.8. Use Short-Lived Containers to Reduce Surface Attack Area

Containers are designed to be ephemeral and lightweight, unlike traditional servers. Comparing container lifespans now versus a year ago, a recent study from Sysdig finds that the number of containers alive for 10 seconds or less has doubled, from 11% to 22%. Likewise, the number of containers that live for five minutes or less more than doubled, from 20% in 2018 to 54% this year. (McKendrick, 2019)

Developers should strive to minimize the number of components in each container and keep all containers as thin as possible. This approach can help reduce the attack surface. In addition, implementing fixes to the base image and pushing that image to the registry for future uses prevents the security vulnerability from being reverted in the environment. This was an issue with legacy hot patching on servers.

Short-lived containers also can minimize the time an attacker is in the environment as any persistence installed by the attacker would be reverted to a clean state. Investing in this practice reduces security risk across the business.

3.2. Known Attacks Where Containers Were Involved

Red Hat reported in their 2022 poll that 46% of respondents worry the most about exposures due to misconfigurations in their container and Kubernetes environments—nearly three times the level of concern over attacks. (Kohgadai, 2022)

In 2022, the most common form of attack against container-based systems was cryptojacking, followed by sensitive data such as API tokens and passwords that were baked directly into the container image, according to Sysdig. Most of these attacks are mainly related to Russia's war against Ukraine.

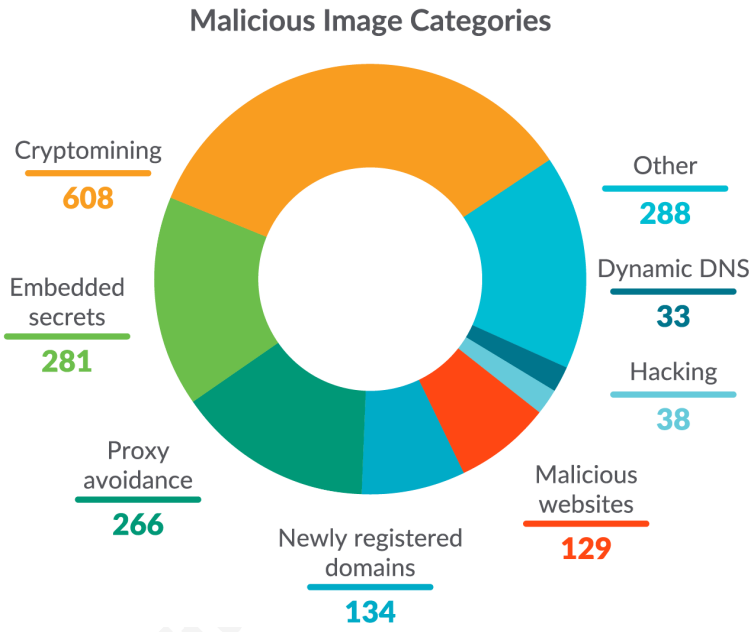

Figure 12. Breakdown of Malicious Container Images

Unfortunately, nowhere in any of the main container repositories (e.g., Docker Hub) does it annotate which containers were compromised or malicious. The only indications of a vulnerability for Docker Hub appeared in the form of a specific security check to ensure that the software "log4j" was not vulnerable. A label was placed on the container to denote the result of the security check. It should also be noted that public data could not be found that tied hacking groups specifically to exact container attacks.

3.3. Top Containers Used by Applications

Docker Hub has exposed its metrics for the most downloaded images to better understand which containers are most used and the security risk exposed by using them. This data is used to understand the risk of the top 25 most downloaded images.

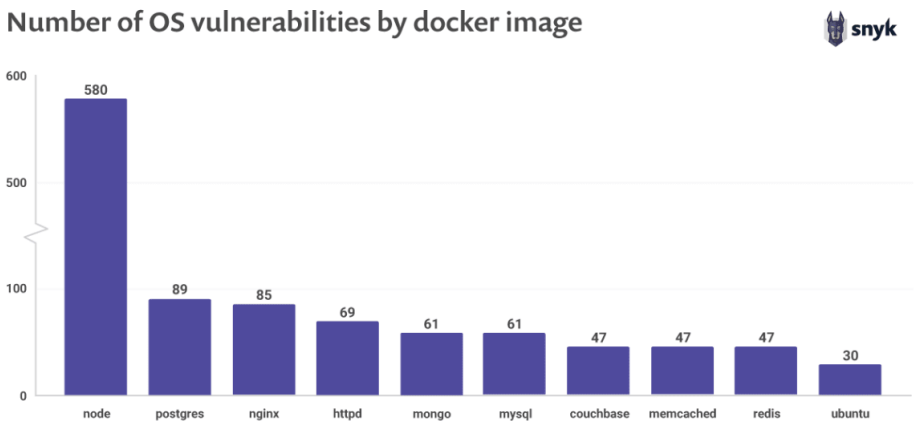

Figure 13. Known Vulnerabilities Among Top 10 Downloaded Repositories

Snyk conducted a 2019 poll and found that open-source maintainers want to be secure, but 70% lack and stated they need strong security knowledge. This same poll found that 44% of respondents had never run a security audit. The number is considerably lower, with 26% of users stating they do not audit their source code.

4. Recommendations and Implications

Developers and container builders have come a long way in improving the security of the containers used today. However, they still need solutions to help facilitate security risk and threat assessments against the configuration of their images and enable developers to make informative security decisions when selecting an image for use in their environment.

The first step in empowering the developers to make more informative security decisions around the images they use is to bring stronger telemetry around the security risks their containers expose. To do this, it will be instrumental to map pieces of the container image back to MITRE ATT&CK Framework with a focus on the container matrix. Doing this programmatically will enable automation for image scanning and produce a result that could be baked into a native UI such as Docker Hub.

The next step in this process focuses on bringing in the image components so a developer can see what makes up the container they plan to run. This work can be used to build signatures around malicious and benign processes.

Brandon Helms, bhelms85@gmail.com

Lastly, improving the developer's knowledge of SSDLC. This is the hardest part because it is fundamental to the success of reducing risk across an organization. While this research does not touch on the training aspect, it is the training aspect that will reduce the security risk.

5. Conclusion

5.1. What is Missing from the Research

During the research, it became extremely difficult to differentiate cyber-attacks based on the tech stack the attackers targeted and as such, finding information on specific attacks against containers was extremely lacking. It was also to derive the attack group used in container-based attacks, which required the research to focus more on aligning container security to the MITRE ATT&CK framework for containers. By shifting to this model, the research was able to continue to the focus on security risk from the perspective of a threat actor.

Secondly, during the creation of Container360, it was determined that there were multiple sections of the MITRE ATT&CK framework for containers that could not be evaluated at a container image level without assessing the orchestration platform that the container image was running in. Seeing that this research excluded orchestration platforms, these sections were left empty and focused on the networking portions and how data was loaded into the container using an orchestration platform.

Lastly, the research had to focus on a small subset of images instead of a larger, multiple application grouping of applications. This was due to each image needing multiple API calls to be made to Docker Hub and Docker Hub rate limiting the requests. Scaling to less than 50 images kept the application from hitting an API rate limit placed by Docker Hub.

5.2. Conclusion

Containers have become the next step in infrastructure evolution, but they will not be the last. While this advancement has increased velocity and standardization for developers, it has also introduced new security risks for organizations that are being

actively attacked by cybercriminals. The use of containers by developers continues to increase, and it is reported that 92% of companies are using containers in their environments. (CNFC Survey, 2020). It is expected that this percentage will continue to increase. With all the images publicly available, it is reported that 85% of containers running in corporate environments have at least one known security vulnerability, and three-quarters of those corporations have at least one known vulnerability of high or critical. (Staff, 2022)

While software vulnerability tooling exists and continues to improve on false positive reduction, there still appears to be no tooling around threat intelligence for containers. This creates a gap in security assessments for organizations and prevents them from having a complete picture of the risks introduced by a container. While this research has supported the statement that containers are creating a false sense of security for organizations primarily due to legacy developer practices and a lack of observability around security from the attacker's perspective, it was tough evaluating a container based solely on the container image itself. This is because while a container can house vulnerabilities or configuration issues, the orchestration platform can mitigate many of these controls based on the orchestration platform's configuration.

The most valuable actions that can be done to reduce the security risk of the container image includes making the image immutable at runtime, packaging only the bare minimal software to run an application, using images that do not come native with shells, expose only services needed to run the application publicly, and remove sensitive information from being stored in the container image. These five items can be attributed to the attacks researched during this research which would have prevented the attack from occurring or significantly decreased the impact of the attack.

This research has also produced a basic prototype that could be streamlined into popular container registries that conveys essential threat intelligence around the container image and uses quantifiable information to allow for informative decision-making on selecting and building container images.

Brandon Helms, bhelms85@gmail.com

References

Ashutosh, C. and Rajewar. R. (2022, January 28). *Attackers cryptojacking Docker images to mine for Monero. Unit 42.* Retrieved March 15, 2023, from https://unit42.paloaltonetworks.com/cryptojacking-docker-images-for-mining-monero/

Armstrong, J. (2022, August 23). *What is container security? | Container Security in 2022.* Retrieved October 16, 2022, from https://snyk.io/learn/container-security/

Clark, M. (2022, September 28). *Sysdig 2022 Threat Report: Cloud-native threats are increasing and maturing.* Sysdig. Retrieved October 16, 2022, from https://sysdig.com/blog/2022-sysdig-cloud-native-threat-report/

Cowperthwaite, A. (2022, July 25). *A CISO's Guide to Container Security: Understanding Vulnerabilities & Best Practices*. Kroll. Retrieved October 16, 2022, from https://www.kroll.com/en/insights/publications/cyber/ciso-guide-container-security

Crippa, F. S. (2021, December 7). *Understanding Container Images, Part 1: Image Layers*. Cisco Blogs. Retrieved October 16, 2022, from https://blogs.cisco.com/developer/container-image-layers-1?ccid=

Corrigan, A. (2022, August 15). *Top 8 container registries.* Octopus Deploy. Retrieved October 16, 2022, from https://octopus.com/blog/top-8-container-registries

DevOpsTV. (2022, March 15). *Container Security Best Practices for Zero Days* [Video]. YouTube. Retrieved October 16, 2022, from https://www.youtube.com/watch?v=w8y7ppEnLGg

Docker Security Best Practices: A Complete Guide • Anchore. (2022, May 24). Retrieved October 16, 2022, from https://anchore.com/blog/docker-security-best-practices-a-complete-guide/

Gunjan Patel. (2019, December 17). *Container Security Best Practices* [Video]. YouTube. Retrieved October 16, 2022, from https://www.youtube.com/watch?v=yctlTHCDtFU

Brandon Helms, bhelms85@gmail.com

Hernandez, Carlos R. (2021), *Best Practices for Securing and Hardening Container Images*. Retrieved October 16, 2022, from https://tanzu.vmware.com/developer/guides/security-best-practices/

Iradier, Á. (2022, October 14). *Container security best practices: Comprehensive guide*. Sysdig. Retrieved October 16, 2022, from https://sysdig.com/blog/container-security-best-practices/

Jones, E. (2021, July 21). *The evolution of a matrix: How ATT&CK for Containers was built*. Microsoft Security Blog. Retrieved October 16, 2022, from https://www.microsoft.com/security/blog/2021/07/21/the-evolution-of-a-matrix-how-ATT&CK-for-containers-was-built/

Lemos, R. (2020, April 7). *Misconfigured Containers Again Targeted by Cryptominer Malware*. Dark Reading. Retrieved October 16, 2022, from https://www.darkreading.com/attacks-breaches/misconfigured-containers-again-targeted-by-cryptominer-malware

Matrix - Enterprise | MITRE ATT&CK®. (n.d.). Retrieved October 16, 2022, from https://attack.mitre.org/matrices/enterprise/containers/

McKendrick, J. (2019, November 8). *Most technology containers live less than five minutes, and lifespans are getting even shorter*. Retrieved October 16, 2022, from https://www.zdnet.com/article/technology-containers-short-lifespans-are-getting-even-shorter/

MITRE ATT&CK®. (n.d.). Retrieved October 16, 2022, from https://attack.mitre.org/

Ms. Smith (2018, February 20). *Hackers exploit Jenkins servers, make $3 million by Mining Monero*. CSO Online. Retrieved March 15, 2023, from https://www.csoonline.com/article/3256314/hackers-exploit-jenkins-servers-make-3-million-by-mining-monero.html

Ovens, S. (2021, November 23). *Building a Linux container by hand using namespaces*. Enable Sysadmin. Retrieved October 16, 2022, from https://www.redhat.com/sysadmin/building-container-namespaces

Sahu, A. (2022, March 24). *Top 10 Container Security Best Practices* - FAUN Publication. Medium. Retrieved October 16, 2022, from https://faun.pub/top-10-container-security-best-practices-317d8070b9a9

Santos, M. D. L. (2020, August 10). *Container Security Tips and Best Practices*. Threat Stack. Retrieved October 16, 2022, from https://www.threatstack.com/blog/container-security-tips-and-best-practices

Staff, V. B. (2022, January 28). *Report: 75% of containers found to be operating with severe vulnerabilities*. VentureBeat. Retrieved March 15, 2023, from https://venturebeat.com/security/report-75-of-containers-found-to-be-operating-with-severe-vulnerabilities/

Tafelski, M. (2022, February 22). *Container Security Best Practices for Better Apps*. Retrieved March 15, 2023, from https://www.trendmicro.com/en_us/devops/22/b/container-security-best-practices.html

Tal, L. (2021, November 14). *81% believe developers should own security, but they are not well-equipped*. Snyk. Retrieved October 16, 2022, from https://snyk.io/blog/81-believe-developers-should-own-security-but-they-arent-well-equipped/

Tal, L. (2022, October 5). *The top ten most popular docker images each contain at least 30 vulnerabilities*. Snyk. Retrieved October 16, 2022, from https://snyk.io/blog/top-ten-most-popular-docker-images-each-contain-at-least-30-vulnerabilities/

Kohgadai, A. (2022, May 17). *The State of Kubernetes Security in 2022*. Retrieved October 16, 2022, from https://www.redhat.com/en/blog/state-kubernetes-security-2022-1

Tigera. (2022, September 15). *What Is Container Security? 8 Best Practices You Must Know*. Retrieved October 16, 2022, from https://www.tigera.io/learn/guides/container-security-best-practices/

Toulas, B. (2022, November 24). *Docker hub repositories hide over 1,650 malicious containers*. BleepingComputer. Retrieved March 15, 2023, from

https://www.bleepingcomputer.com/news/security/docker-hub-repositories-hide-over-1-650-malicious-containers/

Venkat, A. (2022, September 28). *Cryptojacking and DDoS attacks increase in container-based cloud systems*. CSO Online. Retrieved October 16, 2022, from https://www.csoonline.com/article/3675368/cryptojacking-ddos-attacks-increase-in-container-based-cloud-systems.html

Vizard, M. (2022, September 28). *Sysdig Report Reveals True Cost of Container Security Breaches*. Container Journal. Retrieved October 16, 2022, from https://containerjournal.com/features/sysdig-report-reveals-true-cost-of-container-security-breaches/

Walker, A. (n.d.). *The Best Docker Repositories, Apps, and Images in Docker Hub in 2019*. Retrieved October 1, 2022, from https://learn.g2.com/best-docker-containers-repository

Appendix A: Container360 Application

Container360 is still under active development and the installation, layout, and analytics are susceptible to change. Reference the GitHub link supplied for the latest.

A.1 Purpose

During the investigations and analysis of containers, many steps required manual code execution. To speed this up and automate the tests, it was determined that an application needed to be created to house all this logic. This creation allows users to investigate images and store the results in a repeatable format that could be viewed later more rapidly. After seeing these insights in a single place, the next step included adding a basic alerting strategy and aligning it to a common framework.

A.2 Installation

Container360 is an open-source platform in active development. The latest documentation can be found at https://github.com/Cr0n1c/Container360. This application was built on macOS using the m1 processor but should run on any system running Python 3.8 or later with internet access. Once the prerequisites are installed, and the application is started per the "README.md" file, the application should be accessible via http://localhost:8000/search.

```
*[main][~/repos/container360]$ uvicorn main:app --debug
INFO:     Will watch for changes in these directories: ['/Users/brandon/repos/container360']
INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to quit)
INFO:     Started reloader process [20361] using StatReload
INFO:     Started server process [20363]
INFO:     Waiting for application startup.
INFO:     Application startup complete.
INFO:     127.0.0.1:64160 - "GET /image?image_uuid=35742f8156e3b1d98f51c7a450f4dbff5a98bac8 HTTP/1.1" 200 OK
```

Figure 14. Container360 Startup

A.3 Searching Containers

This application relies on a Docker configuration and, by default, will use Docker Hub as the default image repository. To search for a new or previously scanned image, use the "Search" input box at the top of any page.

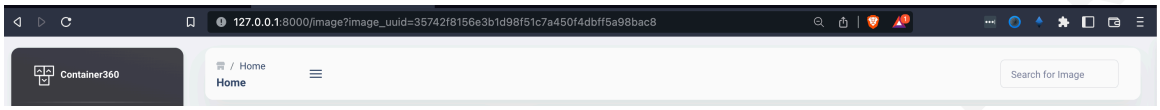

Figure 15. Container360 Header

A.4 View Container Insights

After an image has been scanned using the search, the results will be viewable in the "Image Viewer."This page is set up to give a single-pain view of a container image from a security perspective. This view is divided into three main areas: metadata, MITRE mapping, and insights.

The metadata section contains data relevant to the image. This includes the vendor, image tags, image digest, description, and Container360's unique id for this image scan. Next, the MITRE mapping section uses a signature-based approach to identify potential issues with the container. Lastly, the insights section contains the basic information about the processes, image-building process, and network connections related to the image at runtime. These three views give a holistic view of the container image makeup.

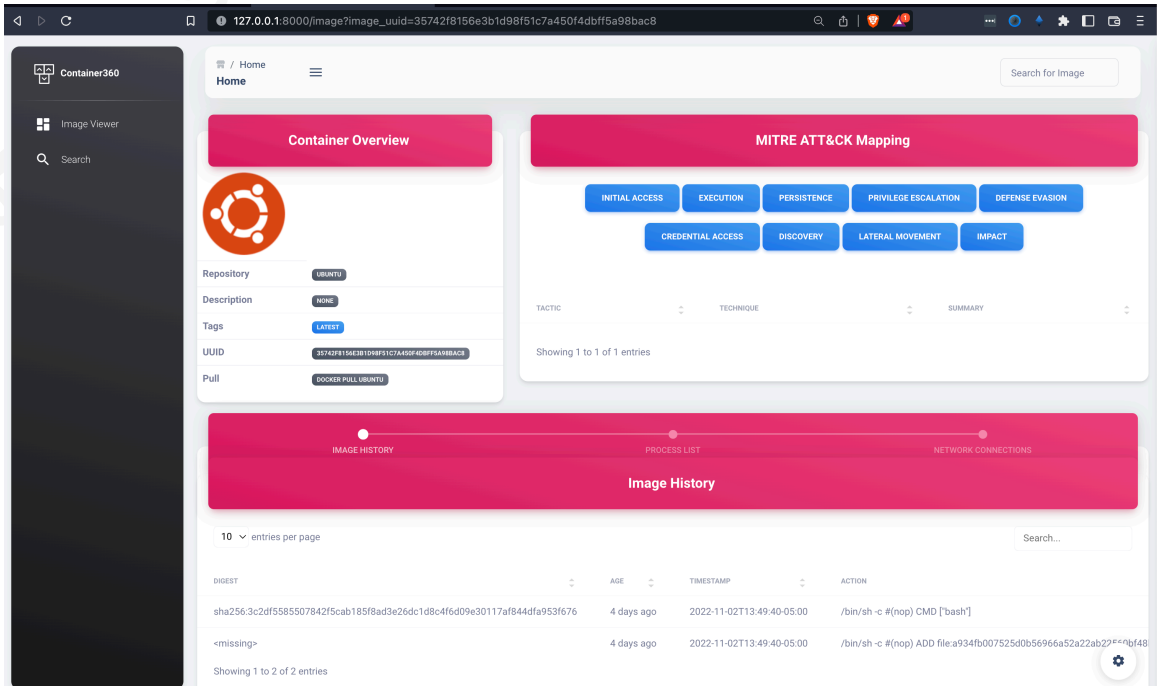

Figure 16. Container360 Image Viewer

A.5 Image Investigator View

If a user wants to find commonalities or uniqueness based on the current images that have been scanned, they can use the "Search" tab. This will present the process history, network connection history, and image build for all images in a user's inventory with free-text search enabled.

A.6 Contribution

To contribute to this project, check out the repository on GitHub and commit any changes back to the project.

GitHub: https://github.com/Cr0n1c/Container360

Brandon Helms, bhelms85@gmail.com

Appendix B: Test Group Configuration

The container sample used for testing focused on using a widespread application in business environments, Nginx. Nginx is a reverse proxy application that allows users and other applications to interact with a single application via HTTP(S), allowing developers to route that traffic to one or more internal applications.

B.1 Sample Group

The sample group was taken from https://hub.docker.com/search?q=nginx using the first three (3) pages of container images, ensuring only one (1) container image per registry. All data analysis came from this sample group on 2023-01-20, and it should be noted that the "latest" image tags are not immutable and are subject to change. A detailed description of each image can be found: https://raw.githubusercontent.com/Cr0n1c/Container360/tests/image_pull_history.txt.

Registry/Repository	Tag
antrea/nginx	1.21.6-alpine
bitnami/nginx	latest
bitwarden/nginx	latest
centos/nginx-110-centos7	latest
circleci/nginx	0.3
clearlinux/nginx	latest
continuumio/nginx-ingress-ws	1.5.2
corpusops/nginx	latest
docksal/nginx	latest
droidwiki/nginx	latest
drud/nginx-base	v0.29.1
gluufederation/nginx	4.5.0-4
intel/nginx	latest
kasmweb/nginx	latest
lancachenet/ubuntu-nginx	latest

Brandon Helms, bhelms85@gmail.com

linuxserver/nginx	latest
mailu/nginx	latest
nginx	latest
nginxcontrib/nginx	latest-ubi
nginxinc/nginx-unprivileged	latest
nginxpro/nginx	latest
nginxpro/nginxpro	latest
objectscale/nginx	3.9.0.0-1569.83e26ae3
okteto/nginx	alpine
privatebin/nginx-fpm-alpine	latest
rancher/nginx	ingress-nginx-1.5.1-hardened2-amd64
rapidfort/nginx	latest
rasa/nginx	latest
semoss/nginx2	latest
treehouses/nginx	latest
ubuntu/nginx	latest
uselagoon/nginx-drupal	latest
vmware/nginx	1.11.5-patched
webdevops/nginx	latest
wodby/nginx	latest

B.2 Condensed Results

All results from automated testing were repeated using manual testing to ensure the accuracy of the findings. A condensed version of the results for all images can be found https://raw.githubusercontent.com/Cr0n1c/Container360/tests/results.xlsx (results.xlsx).