Introduction

As more organizations continue their migration to the cloud, the adoption of **microservices and containerized workloads** is rapidly reshaping how infrastructure is deployed, scaled, and secured. This shift introduces new challenges , especially in how we **detect and monitor malicious activity** within virtualized environments. Traditional host-based detection alone is no longer sufficient.

In this write-up, I walk through a real-life scenario where a **malicious Docker container was executed on a Linux endpoint** following a brute-force attack. This investigation serves as a microcosm of a much larger challenge: while analyzing a single container can be manageable, **monitoring hundreds or thousands of containers across enterprise-scale infrastructure requires dedicated visibility into the virtualized layers themselves** , including container orchestration platforms, cloud APIs, and container runtime behavior.

This incident illustrates how attackers can exploit exposed or misconfigured environments, and why securing virtualized infrastructure is a growing priority in modern security operations.

Let's find the alert details below

EventID :253
Event Time : Apr, 29, 2024, 09:42 AM
Rule : SOC253 - A Malicious Docker Container Executed
Level : Incident Responder
Source Address : 172.16.20.36
Host : DockerBox
Alert Trigger Reason : A Malicious Docker Container Executed
Alert Trigger Command : docker create --name test_container ubvntu/utnubu
Hash :16805a9f91e9ead7f803e12ddb4f2ef6432a21887dca4e991bc6fdd2f58ebb0c
L1 Note :
When I check the relevant command, I see it in "history" on the system. However, I could not confirm whether the command belongs to the attackers or whether it was executed within the scope of business processes. In addition, minutes before the alert, I saw Brute Force attempts from the IP "89.39.107.194" towards the system.

Summary of Observations

Based on the alert triggered by SOC253 — “A Malicious Docker Container Executed” — at 09:42 AM on April 29, 2024,

1. A suspicious Docker container was created on the host `DockerBox` using the command `docker create --name test_container ubvntu/utnubu`. The image name appears to be a **typosquatted variation of** `ubuntu`, which strongly suggests deceptive intent or an attempt to execute an untrusted image.
2. The command was found in the system's `.bash_history`, indicating it was run locally on what appears to be a **Linux endpoint**, given the command syntax and environment context.

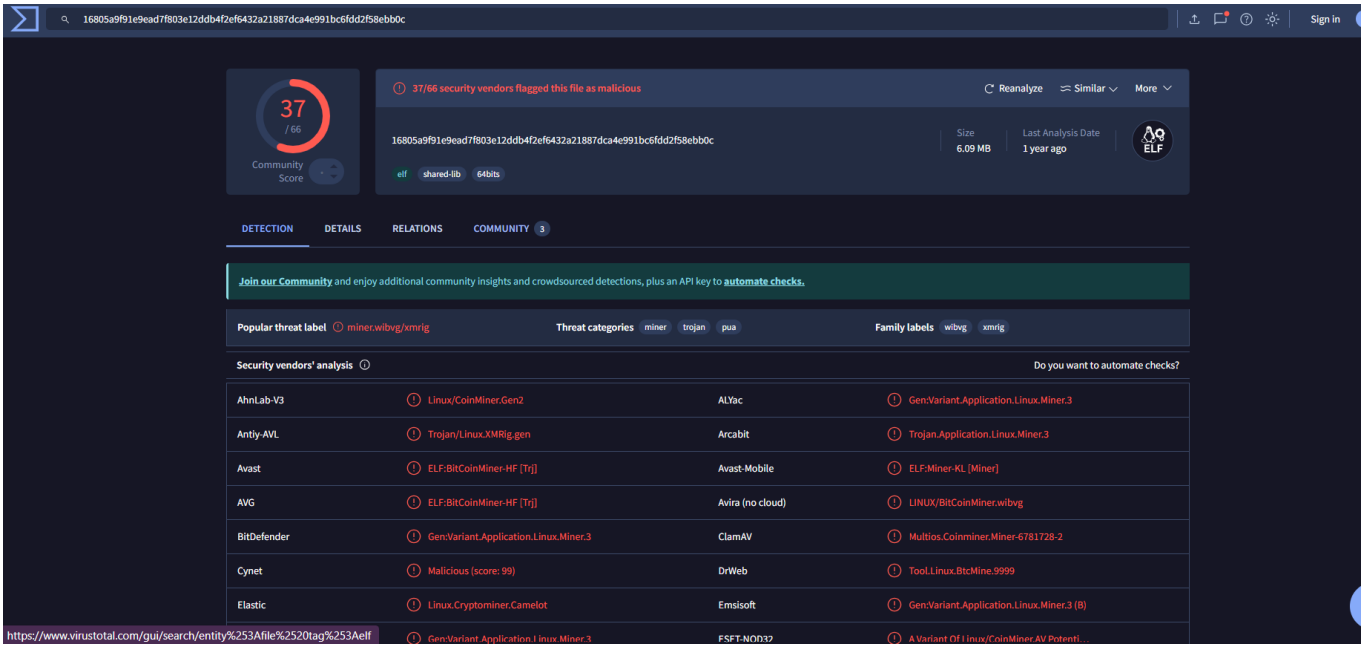
TP or a FP

At the start of the investigation, the main question was the classic SOC dilemma: **is this a true positive or just another false alarm trying to ruin someone's coffee break?** Our L1 analyst wasn't quite sure the Docker command showed a suspicious image being created, but it wasn't immediately clear if it was part of a legitimate business process or someone going rogue with containers. Digging deeper, we noticed that **minutes before the alert**, there were **brute-force login attempts** from IP `89.39.107.194`. A quick **threat intel lookup** confirmed that this IP has a history of bad behavior (the kind that makes blocklists nervous). On top of that, the **SHA256 hash of the image** had already made the rounds on **VirusTotal**, getting flagged by multiple engines not exactly the résumé of a trusted Docker image. With all that in mind, we were able to reassure our L1: **yes, this is definitely a true positive**, and no, it's not part of any normal business activity unless your business involves downloading typosquatted malware in containers.

abuseipdb threat intel

Reporter	IoA Timestamp (UTC)	Comment	Categories
rtbh.com.tr	2025-10-28 20:09:37 (4 days ago)	list.rtbh.com.tr report: tcp/0	Brute-Force
rtbh.com.tr	2025-10-27 20:09:35 (5 days ago)	list.rtbh.com.tr report: tcp/0	Brute-Force
venus.launch.bz	2025-10-26 20:06:47 (6 days ago)	(wpscan) WordPress probe detected from 89.39.107.19 4 (NL/The Netherlands/89-39-107-194.hosted-by-wor ... show more	Hacking
cmbplf	2025-10-26 08:45:26 (6 days ago)	913 requests with url.path */xmlrpc.php	Brute-Force, Bad Web Bot
TPI-Abuse	2025-10-26 02:32:26 (6 days ago)	(mod_security) mod_security (id:225170) triggered by 8 9.39.107.194 (89-39-107-194.hosted-by-worldstr ... show more	Brute-Force, Bad Web Bot, Web App Attack
Anonymous	2025-09-03 22:10:23 (1 month ago)	Ports: 80,443; Direction: 0; Trigger: LF_CUSTOMTRIGG ER	Brute-Force, SSH
TPI-Abuse	2025-09-03 21:59:03 (1 month ago)	(mod_security) mod_security (id:225170) triggered by 8 9.39.107.194 (89-39-107-194.hosted-by-worldstr ... show more	Brute-Force, Bad Web Bot, Web App Attack
Global Cyber Police	2025-07-27 18:19:43 (3 months ago)	Malicious bot activity detected: Hitting honeypot page (2 00 OK with 258/259 bytes sent).	Port Scan, Brute-Force, Web App Attack
Global Cyber Police	2025-07-27 18:19:43 (3 months ago)	Malicious bot activity detected: Hitting honeypot page (2 00 OK with 258/259 bytes sent).	Port Scan, Brute-Force, Web App Attack
Anonymous	2025-07-10 05:14:03 (3 months ago)	Ports: 80,443; Direction: 0; Trigger: LF_CUSTOMTRIGG ER	Brute-Force, SSH

virustotal threat intel



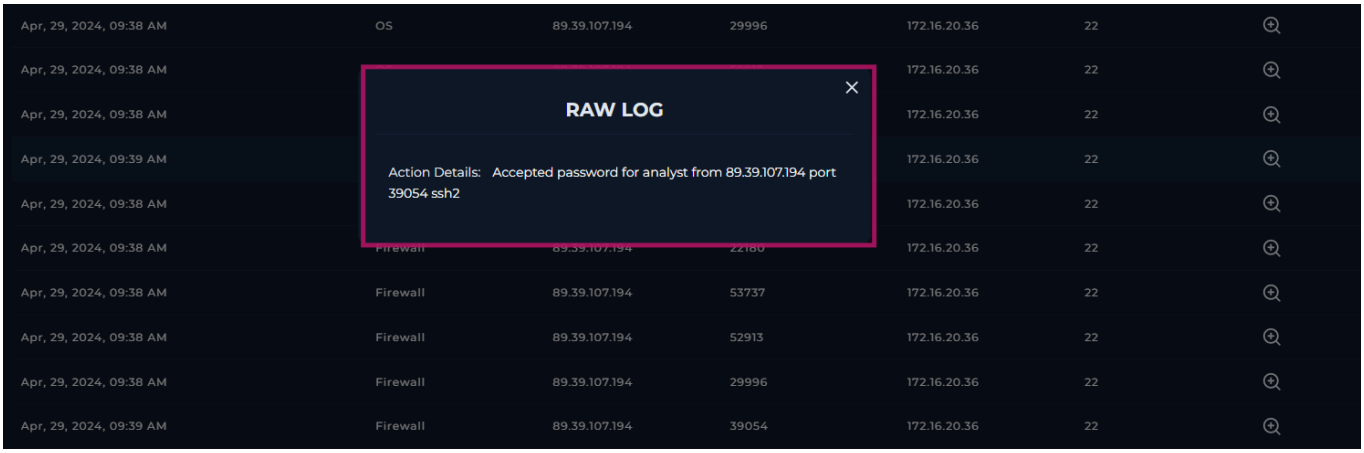
Launching into investigation

Aiding our decision that the alert is a TP, the **SIEM logs told a more compelling story**: they showed a clear sequence of **brute-force login attempts** from IP `89.39.107.194`, followed shortly by a **successful login** , a red flag in any environment. A threat intel lookup confirmed that the IP was tied to known malicious activity. To top it off, the **SHA256 hash of the Docker**

image used in the command (`ubvntu/utnubu`) had been flagged across multiple engines on **VirusTotal**, making it clear that this was no harmless container experiment. With that evidence stacked, we were able to confidently assure our L1: **yes, this is a true positive**, and **no, deploying malware via typo'd Docker images is not in our business scope** , unless, of course, we've pivoted into cybercrime as a service (spoiler: we haven't).

bruteforce attacks with failed passwords

A successfull login



With confirmation that this was a **true positive**, we pivoted to the endpoint to investigate **what the attacker did after gaining access**. Reviewing the command history on the compromised Linux host (`DockerBox`), we observed a clean, step-by-step sequence of activity , the kind you'd expect from someone **getting their bearings in an unfamiliar environment**. Here's how it unfolded:

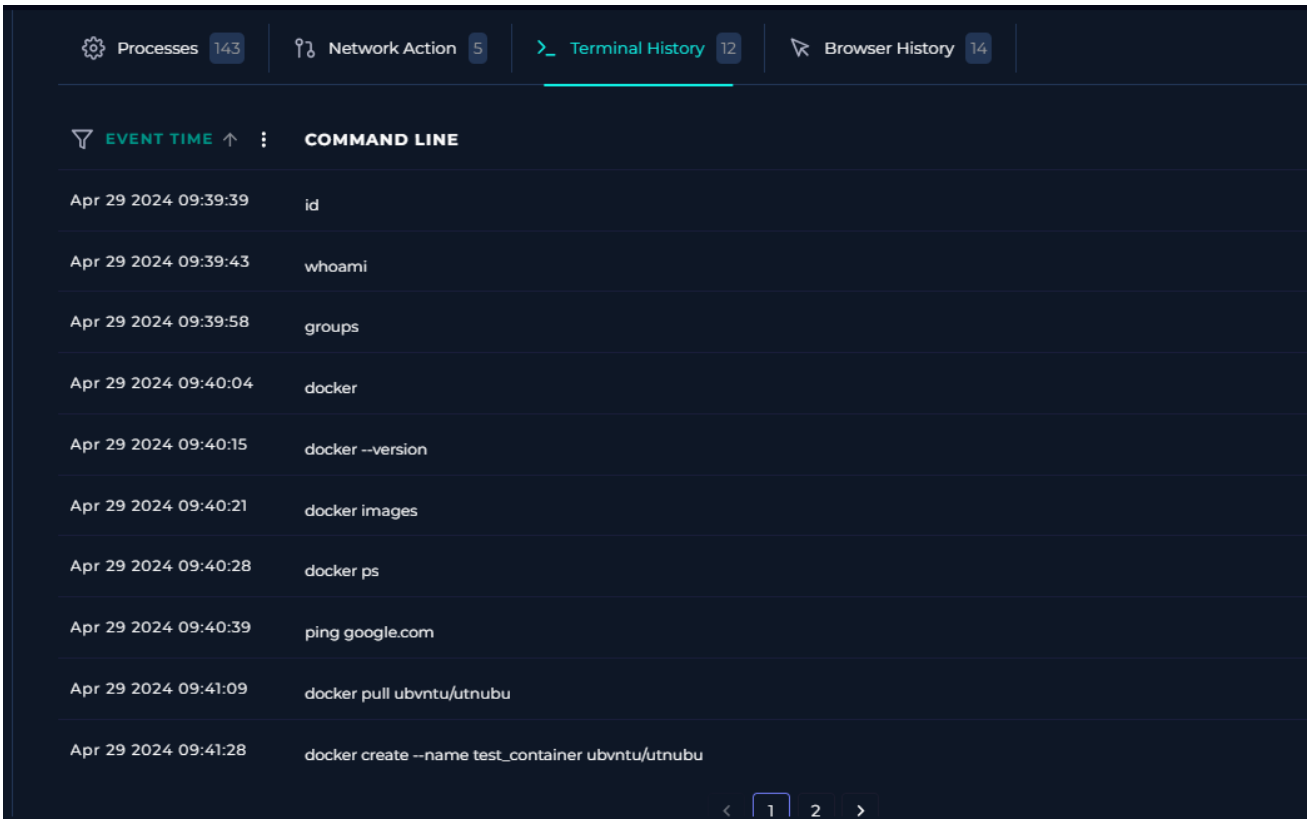
The attacker starts with basic **system reconnaissance**. `id` , `whoami` , and `groups` help determine **what user they're logged in as**, what privileges they have, and what system groups (like `docker`) they might belong to. This is typical "where am I and what can I do?" behavior.

Now that they know they have access, they **probe the Docker environment**.
They check if Docker is installed, what version it is, whether there are pre-existing images or containers running , likely to confirm whether they can **leverage Docker as an execution platform**.

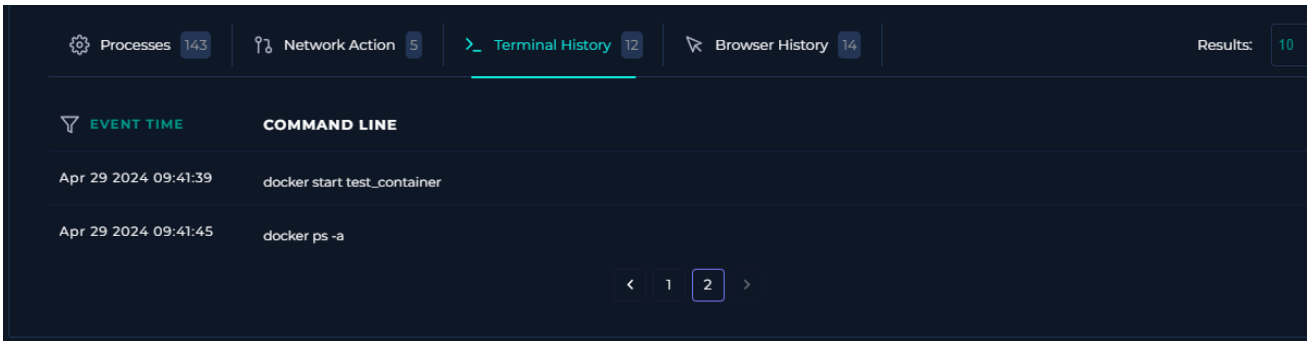
A quick connectivity check : probably to confirm **internet access** for pulling images or fetching remote payloads. This is also a subtle way of confirming **DNS and outbound access**, which many attackers test before continuing.

And now we get to the core of the attack:

- The attacker pulls a suspicious, **typosquatted image** (`ubvntu/utnubu`) from Docker Hub or a similar registry.
- Then they **create and start a container** using this image ,potentially running malware, reverse shells, miners, or persistence mechanisms inside it.
- The final `docker ps -a` confirms that the container is running and may also serve as a way to verify it started successfully. *terminal history*



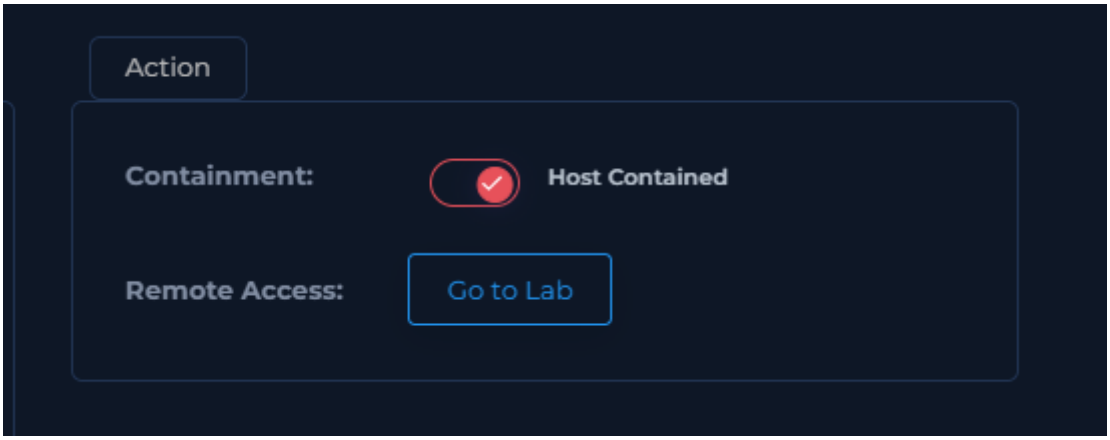
terminal history



Containment

Since this was a single host running Docker without orchestration, containment was straightforward. We first **isolated the host from the network**, either by physically disconnecting it or applying outbound firewall restrictions, to cut off any potential command-and-control (C2) traffic. The **malicious container was then stopped**, but preserved for forensic analysis (rather than deleted outright).

However, this would have been significantly more complex if the environment were a **production cluster (e.g., Kubernetes)**. In that case, we wouldn't simply kill a container containers could respawn automatically due to orchestration logic. We'd have to **cordon the affected node**, possibly **quarantine the namespace**, and apply strict `NetworkPolicies` to cut off traffic to and from the malicious pod. Additionally, if the attacker had used a service account, we'd need to immediately **revoke credentials or tokens** used across other pods or services.



Eradication

On the standalone Docker host, eradication focused on understanding how the attacker got in. SSH logs and shell history were analyzed to trace the intrusion path. If SSH was exposed publicly or password authentication was used, we transitioned to **key-based authentication** and enforced firewall-based IP restrictions. If the attacker used a vulnerable or trojanized **base image**, we checked all locally stored images and rebuilt any that were not verified from a secure source.

In contrast, had this been a cluster, eradication would involve **reviewing imagePull policies**, validating **container registries**, rotating **cluster-level secrets**, and auditing **Role-Based Access Control (RBAC)** permissions to ensure the attacker hadn't leveraged excessive privileges. We'd also likely **rotate compromised node credentials** and rebuild affected nodes from golden images.

Recovery

With the standalone Docker host, recovery was relatively fast. Once the malicious container was removed and the host verified as clean, we deployed a new container using a **known-safe base image**. This avoided downtime, as containers can be restored within seconds.

In a clustered environment, however, recovery would be **pipeline-driven**. Affected deployments would be **recreated using version-controlled IaC (Infrastructure as Code)** pipelines, with **automated security checks and container admission policies** enforced. Nodes might be removed from the cluster, rebuilt, and rejoined only after passing integrity checks. Cluster-wide

configurations like **network segmentation, pod security policies**, and **image scanning** would play a crucial role in ensuring safe recovery.

Lessons Learned

This incident highlighted a critical truth: the presence of a malicious container was not the root of the problem, but a symptom of a deeper compromise. The attacker had likely gained unauthorized access through SSH, enabling them to stop a legitimate container and launch a malicious one in its place. While containment was executed effectively in this standalone Docker setup, it became clear that future responses must prioritize root cause analysis over surface-level remediation.

One key takeaway for both staff and management is the importance of treating container incidents as part of a larger intrusion lifecycle. Rather than reacting to what's visible, teams must examine how the attacker gained access, what persistence mechanisms may be in place, and whether lateral movement is underway. Management should also emphasize cross-team coordination among incident response, DevOps, and infrastructure stakeholders, ensuring there is a shared understanding of container deployment pipelines and escalation workflows. In orchestrated environments like Kubernetes, the importance of automated, repeatable incident handling through infrastructure-as-code becomes even more critical.

Several remediation measures can help prevent similar incidents in the future. SSH access should be hardened by disabling password-based authentication, enforcing key rotation, and restricting remote access through cloud-native firewall controls or security groups. Image security must also be enforced at the build level by using signed, scanned container images and validating them with admission controllers or runtime enforcement tools. Host and container visibility should be strengthened by deploying runtime security tools such as Falco or Sysdig, and access control at the container level should be enforced using technologies like AppArmor or seccomp profiles. Shifting toward immutable infrastructure practices i.e where compromised nodes or containers are rebuilt entirely rather than patched, can also significantly reduce the risk of reinfection.

Looking forward, several key indicators and precursors should be continuously monitored. Unusual SSH logins or brute-force attempts from unfamiliar IPs, particularly those targeting root or admin accounts, should trigger immediate alerts. Unscheduled container start events, especially from unsigned or unverified images, are red flags for unauthorized deployment. The abrupt termination of known-good containers, followed by the startup of new, untracked containers, may indicate attacker behavior. Network indicators such as outbound connections to known command-and-control servers, TOR exit nodes, or uncommon ports should be correlated with container activity. Finally, in a clustered environment, anomalous usage of pod service accounts or access tokens especially outside of expected time windows or from non-standard nodes i.e should be closely scrutinized.

By refining visibility, enforcing secure defaults, and preparing environment-specific playbooks, future container security incidents can be detected earlier, contained more effectively, and remediated with minimal disruption.