Docker Forensics for Containers: How to Conduct Investigations

cloud.redhat.com/blog/docker-forensics-for-containers-how-to-conduct-investigations

Michael Foster

Forensics in the age of containers

You’ve seen it countless times in television’s most popular dramas: professional investigators descend on the scene of a crime to meticulously record and analyze every detail and clue before anyone else can disrupt the scene. If the crime appears to be related to other ongoing cases, clues are tacked to the peg board back at headquarters. Only once all the pieces have been assembled do patterns emerge.

Forensics in the realm of container environments are somewhat similar, only that suspicious activity unfolds much faster. Following the trail invariably means piecing together data from multiple systems, and that data needs to be captured with as little modification or disruption as possible.

Containers make digital forensics incredibly complex, as they are scheduled and orchestrated across different hosts according to usage and need. Furthermore, container environments yield enormous amounts of data at high velocity, which is difficult to capture without the right type of instrumentation and tools.

In contrast with virtual machine (VM) environments, containers running on a host share the underlying operating system kernel. There isn’t any hardware isolation at play. There may also be new restrictions to account for, such as network overlay isolation, even within the same host.

The good news is: the IT industry has seen this sort of complexity transition before. When VMs were first introduced, the standard practice was to copy the memory and disk contents directly from the physical server. Multiple VMs may have been transitioned to or from a host machine that was first exploited, or a copy might be running on another bare metal machine. Despite the fact that VMs can be migrated between different data centers, they are considered sufficiently self-contained to provide evidential preservation.

StackRox envisions the same situation with respect to containers. Similar to the transition from physical servers to VMs, there will be nuanced complexities, with associated advantages and disadvantages.

Though not intended to be taken as legal advice, this technically-focused blog post offers up an overview of some of the key elements of digital forensics, along with key considerations for conducting effective forensic investigations in container environments.

Component evidence

Containers differ from bare metal or virtual machines in a number of ways that impact obtaining actionable evidence. At this time there is no default “container snapshot” function available; containers must be captured as a set of distinct components.

File system

Leading container runtimes use a copy-on-write file system. This is a great advantage to forensic acquisition. All of the default system and application files exist within the container image. Any changes since the container started are stored in a separate directory from the original image. Furthermore, any deletion of original files from the image is also recorded.

If you’re using Docker, you may manually explore the copy-on-write diffs inside `/var/lib/docker`. But we can go one step further: Docker will capture all file modifications, since the container was started into a new layer on the copy-on-write file system. Simply commit an existing container, running or not, into a new image: `docker commit $CONTAINER_ID imagename`. Committing a container into a new image does not otherwise modify the container in any way.

Note that committing a container differs from snapshotting a VM. Committing only records file system changes, but not the current process execution state. If you resume a container from a commit, it will need to launch new processes and is not guaranteed to execute any suspicious behavior.

Memory

Containers offer additional benefits in terms of memory isolation. Containers are not a first-class primitive in linux; they are an abstraction on top of multiple subsystems, such as process namespaces and cgroups. Processes running inside a container manage memory the same as any other process. By default, processes inside a container may not interact or interfere with memory controlled by any process outside the container. This provides the following characteristic: if you capture all of the memory for each process in a container, you have captured all allocated memory for that container.

There exist multiple per-process memory dump utilities in linux. The utility `gcore` suspends the process during acquisition but does not modify it; it executes it quickly, and the output format is compatible with YARA. Alternatives include `obdump`, `[memfetch](https://github.com/citypw/lcamtuf-memfetch)`, `gdb`, or even `dd` directly from memory devices.

Shared volumes

Containers are ephemeral. Malware might not be aware of this and write some interesting data to the container's allocated disk space. Ultimately the most sensitive data is likely stored via a volume, typically mounted against persistent online storage. This is where container isolation breaks down.

Mounted volumes rely on a storage backend. By default, that's the host's native file system. Frequently, a distributed or cloud file system is used, such as gluster or Amazon S3. Acquiring and exploring these file systems will require interacting with the file system directly.

It is not recommended to use a container that also mounts this volume to enumerate files – that may tarnish the file access and modification metadata. If you absolutely must access the volume this way, be sure to mount it into the container as 'read-only'.

Identifying the volume mount is simple; one inspects the container. Volumes are identified via the Volume and Mount configuration. Existing examination techniques will be needed for each backing service.

Related microservices

Containers are typically employed within a microservices-based architecture. While great for speed and scalability, this means that suspicious activity will likely involve multiple containers pertaining to multiple services, across various hosts and with access to different data stores.

While investigating a container breach, make note of any data stores or services that this container accessed. Container acquisition is much faster than VM acquisition, but the price we pay is that one may need to acquire evidence from all of these associated containers.

The web of lateral movement across containers will differ according to the sophistication of an attack. It is better to acquire more now than wish you had, later on.

StackRox is an indispensible asset to security teams, as it records network traffic information between individual containers and detects malicious lateral movement. This enables an analyst to track down potentially compromised containers very quickly.

Container escape

There are vulnerabilities and misconfigurations that could allow malware to escape a container. If there is any evidence of suspicious behavior on the host itself, or evidence of this type of malware, it is prudent to image the entire host, whether bare-metal or VM.

Even if you must capture the entire host, organizing the evidence by container can dramatically reduce the investigation effort required. A container may not be fully isolated in some cases, but until there is evidence of an isolation bypass, one may assume that the suspicious behavior is constrained to the container's environment.

Container isolation falls into the following categories:

- **Network isolation**: containers may only contact containers for which they are provisioned on an overlay network (Docker Swarm) or through iptables routing (kubernetes).
- **Process namespacing**: containers cannot see or interact with any other processes running on the same system unless explicitly authorized.
- **File system chroot**: containers may only modify files under which it was explicitly launched or mounted.
- **Device access control**: containers only have access to system devices if explicitly provided.
- **Default seccomp profile**: containers, by default, may not interact with system commands such as shutdown.

Note that all of these default isolation properties may be explicitly bypassed. For example, a Docker container mounting the root of the host file system, or using the host pid namespace will be able to easily escape the container environment, perhaps without leaving behind any indicators. If a container is launched using any of the following flags, host examination is warranted:

- `--cap-add *`
- `--device *`
- `--device-cgroup-rule`
- `--ipc system`
- `--mount /`
- `--pid system`
- `--privileged`
- `--security-opt`

- `--volume /`
- `--volumes-from`

StackRox can alert you if any containers launch with these increased privileges and when the container uses them.

Containers in a forensic environment

At this time, there isn't a formal mechanism for running a captured container. Once they're shut down, even if both file system and memory contents are exported, there is no mechanism for combining the two back into the previous running state. Containers are designed to be ephemeral and thus start processes and allocate new memory, upon initialization.

To that extent, interaction during execution is mandatory, and it is possible to preserve the container and remove it from a production environment. There are a few mechanisms one can use to persist the running container without the container further impacting the cluster in a significant way.

First, containers may be paused at any time. Container execution is completely suspended, memory remains allocated, volumes remain mounted, etc. Any malware currently executing is interrupted.

Second, containers may be quarantined by removing network access or system call privileges. The container processes – and possibly the malware – will continue to execute, but will be unable to send, receive, read, or write any data, or even unmap existing memory. However, this causes errors. Depending on how the processes handle these errors, this may stop the container.

As a container security platform, StackRox is unique in that we offer a comprehensive set of response features that can be automatically executed according to user-defined security policies. Beyond alerting on anomalous or malicious activity – such as lateral movement, or a container with excessive privileges – StackRox will block unauthorized Docker commands, system calls, and quarantine, isolate, or instantly pause compromised or rogue containers.

Once a container is paused or quarantined (if one desires a clean operating environment), it's possible to scale up a cluster. An administrator can add a new node to the system, drain all other containers from this node, and optionally remove it from the orchestrator. At that point, network access may be restricted from the node, and the container can be resumed or switched to a different network.

Conclusion

Even as enterprises move away from monolithic applications to containers and microservices architectures, they still face a wide spectrum of cyber attacks aimed at stealing valuable data, commandeering infrastructure, or causing wanton disruption and destruction. Forensic analysis is a critical capability of any enterprise security program, and it must take into account the characteristics of the significantly different attack surface and environment that containers form. Understanding container ephemerality and isolation is required in order to identify infected containers, capture components, and analyze them safely.

StackRox enables detailed, comprehensive visibility into container activity, and puts a best-in-class set of policy-driven response capabilities in the hands of security practitioners that will dramatically enhance the efficiency and effectiveness of investigations.