Volatility plugin contest 2021 Submissions - Linux namespaces support & Docker plugin

Ofek Shaked & Amir Sheffer

Abstract

The objective of this project is to create a suite of Volatility 3 plugins for memory forensics of Docker containers. To achieve this, we developed improved versions of some of Volatility's core plugins, intending to make them aware of Linux namespaces. Most of these plugins were never ported from Volatility 2, so they were remade to some extent.

It is important to note that we added compatibility for older kernel versions to our plugins. We have tested the following kernel versions: 5.4, 4.4, 3.11, 3.0, 2.6.32.

After improving said core plugins, we used the additional namespace-related information they provide and developed the main plugin for this submission - the Docker plugin.

The Docker plugin has a few options:

- **detector -** When choosing this option the plugin will give the investigator a quick indication about the presence of Docker / Docker containers running on the machine.
- **ps -** When choosing this option the plugin will display a table, similar to *docker ps* command output, that shows the following details about running containers on the machine: container creation time, running command, container-id, is privileged, container process PID.
- **inspect-caps -** When choosing this option a list of running containers will be displayed and the plugin will enumerate the containers' capabilities.
- **inspect-mounts -** When choosing this option a list of non-default mounts will be displayed with information about the associated container, mount paths, and mount options.
- **inspect-networks -** When choosing this option a list of Docker networks will be displayed by their IP segments and the containers that are related to them.

An overview of how the plugins work is supplied below.

Changes/additions to Volatility core plugins

linux.pslist.PsList

This plugin, which was already present in Volatility 3, received an upgrade in the information that is displayed (or returned to another plugin that uses it).

When running without any arguments, the start time of each task is displayed. The start time calculation is based on Volatility 2 which already displays this information.

An optional argument *--nsinfo* was added, which when used, displays information about the namespaces of each task:

- PID in NS - the effective PID of the task, within its PID namespace.
- UTS NS - the UTS (UNIX Time-Sharing) namespace of the task

- IPC NS - the IPC namespace of the task
- MNT NS - the mount namespace of the task
- NET NS - the network namespace of the task
- PID NS - the PID namespace of the task
- USER NS - the user namespace of the task

Another optional argument *--credinfo* was added that displays user and capabilities information:

- Real UID - The real UID in the context of which the task is running
- Real GID - The real GID in the context of which the task is running
- Eff UID - The effective UID which dictates the task's permissions
- Eff GID - The effective GID which dictates the task's permissions
- CapInh - The inheritable capabilities of the task (as a number)
- CapPrm - The permitted capabilities of the task (as a number)
- CapEff - The effective capabilities of the task (as a number)
- CapBnd - The bounding capabilities of the task (as a number)

When using the pslist plugin from another plugin, the *get_task_info* class method is available, which when given a task, returns the above information about the task in the form of a *TaskInfo* object, which contains all information about the task.

linux.pstree.PsTree

The pstree plugin was only adapted to the changes made in pslist. As it inherits from the pslist plugin, the *--nsinfo* and *--credinfo* arguments can be used with it as well.

linux.mount.Mount

The mount plugin is where most of the hard work was done. This plugin, which only existed in Volatility 2, relied on *mount_hashtable* to extract the mounts. This data structure, which as the name suggests, is a hashtable of all mounts and is not the correct way to list mounts on the system.

Linux mounts are defined in the context of the mount namespace they belong to, which may change between different processes (and often does when dealing with containers).

The improved mount plugin can be used in 2 ways. The first method is using the *--all* argument, all mounts are extracted from the *mount_hastable* like in Volatility 2, except that unlike Volatility 2 which inexplicably filtered them, they are not filtered in the updated plugin.

The second method is not using the *--all* argument. In this method, the *--pid* argument is used to supply a list of PIDs. The pslist plugin is then used to retrieve the corresponding tasks. The mount namespace is extracted from each task, which is then used to extract a list of mounts that belong to that mount namespace, using the *list* member of the *mnt_namespace* struct. If *--pid* is not specified, the mounts for PID 1 are extracted.

After extracting a list of mounts, the following information is displayed about each mount:

- Mount ID

- Parent ID - The ID of the parent mount
- Devname - the device name
- Path - the FS path on which it is mounted, as seen from within its mount namespace
- Absolute Path - for mounts that can be accessed from the main mount namespace, this is the path that is used to access them
- FS Type
- Access - read and write access
- Flags - the mount flags, in textual form

If the optional *--sort* argument is used, mounts are sorted by their mount ID.

The mount paths are now calculated by emulating the *prepend_path* function in the Linux kernel. This function requires an FS root for context, which is obtainable from the task to which the mount belongs, as we now have this information (except when running with *--all*, in which case the old method is used).

When using the mount plugin from another plugin, 3 class methods are available. The *get_mounts* function extracts mounts based on the PID filter given (same as running the plugin with *--pid* or with no arguments), and the *get_all_mounts* function extracts all mounts using the *mount_hashtable* (same as running the plugin with *--all*). Both functions generated a tuple of a task and a mount, for context. *get_all_mounts* returns None instead of a task for consistency purposes.

The final method, *get_mount_info*, return all information that is displayed when running the plugin.

linux.file.ListFiles

The ListFiles plugin offers the same functionality as the enumerate_files plugin from Volatility 2. It lists all files recursively under all mounts of the mount namespaces belonging to the tasks supplied by the *--pid* argument (if it is not specified, PID 1 is used).

For each file, the following info is displayed:

- Mount ID - the ID of the mount to which the file belongs
- Inode ID - the ID of the file's inode
- Inode Address - the address of the file's inode
- Mode - the file permissions (as shown when running ls -l)
- UID - file owner UID
- GID - file owner GID
- Size - size in bytes
- Created - created time (epoch)
- Modified - modified time (epoch)
- Accesses - accessed time (epoch)
- File Path - path as seen from within the mount namespace of the mount it belongs to

The following optional arguments may be used:

- *--mount* - show files from the supplied list of mount IDs. If *--pid* was not specified, mounts from all PIDs are extracted before filtering on the specified mounts.
- *--path* - show files whose paths begin with the specified path (for example *--path /etc* will list all files under the /etc directory, including the /etc directory itself).
- *--uid* - show files whose owner is in the specified list of UIDs
- *--sort* - sort files by path

linux.ifconfig.Ifconfig

The ifconfig plugin is mostly a port of the Volatility 2 ifconfig plugin, except for some of the network interface information that is extracted.

The first change is the *get_net_devs* class method, which based on automatic detection of the kernel version, selects between the 2 network device extraction methods implemented in *_get_devs_base* and *_get_devs_namespaces*.

After extracting all net devices, info from them is extracted using the *get_net_dev_info* class method. This method, in addition to Volatility 2's ifconfig plugin, extracts the IPv6 address of each network device (if applicable).

Also, the method of extracting the MAC address was changed - now the *dev_addrs* field of the *net_device* struct is used to obtain the MAC address of the device.

The final change from the Volatility 2 version, is that the subnet mask is displayed next to the IPv4 and IPv6 addresses in CIDR (slash) notation.

Changes made in the Volatility framework

To accommodate all of the plugin changes and additions mentioned above, several additions to the Linux symbol extensions in the Volatility framework were made.

These additions include, but are not limited to:

- The *VolTimespec* class, which is used in Volatility 2 for keeping time-related info, along with the *get_start_time* method in the *task_struct* extension class
- Methods in the LinuxUtilities class that retrieve time-related information
- The *prepend_path* method in the *LinuxUtilities* class, which as mentioned above, calculates an FS path as it is calculated in the Linux kernel
- Methods in the *task_struct class* that retrieve namespace-related information
- An extension class for *hlist_node*, which allows simple iteration over lists like with *list_head*
- An extension class for *nsproxy*, with methods to retrieve namespace objects in a kernel version agnostic way
- A *GenericNamespace* class, along with extensions classes for the different Linux namespace structs (that inherit from *GenericNamespace)*. The *GenericNamespace* class provides a method to retrieve the namespace number (*inum*) in a kernel version agnostic way using the *get_inum* method.
- An extension class for *kernel_cap_struct* that allows conversion of the capabilities value to a single integer

- An extension class for *net_device* which allows for kernel version agnostic extraction of *ip_ptr* and *ip6_ptr*, which in older kernel versions require casting from void pointer to the appropriate struct

Docker plugin

detector

The `detector` option shows a table with these boolean values that can help an investigator quickly detect the presence of Docker daemon or running containers:

- Docker interface - Searches for Docker bridge interface by its name and its MAC address. The Docker interface has a constant name - docker{int} and a constant first 4 bytes in the MAC address - 02:42.
- Docker veth - For each running container, a virtual network interface is created with a constant first 4 bytes in the MAC address - 02:42 and a generic name - eth0.
- Mounted overlay FS - Docker uses a COW FS named overlay. The plugin searches for overlay FS mounted under */var/lib/docker*.
- Containerd-shim is running. Containerd-shim is the parent process of each Docker container that takes care of the container lifecycle. The plugin searches for such processes in pslist output.

ps

The `ps` option is an emulation of Docker's `ps` command.

It enumerates containers processes as *containerd-shim* sub-processes. As written above, each running container must be a sub-process of the *containerd-shim* process that handles a few things in the container's lifecycle such as I/O, logs redirection, `docker attach` functionality and more.

The *containerd-shim* process must live as long as the container lives so we can depend on it to enumerate the container's processes.

It prints a table with these values:

- Creation time - Container's process starting time. Taken from our pslist plugin.
- Command - Container's starting command. Taken from the comm property of the task struct.
- Container-id - This value is a unique value that Docker gives each container when it starts it. Docker creates a directory, named after container-id under */var/lib/docker/overlay(2)/container-id/merged* which is the root FS of the container. For each container process, the plugin enumerates the process' mount points using our improved version of the linux.mount plugin.
  It searches for this directory and extracts container-id from its path.
- Is privileged - The plugin checks if the process capabilities of the container are equal to *init* process (PID 1) which indicates that all capabilities are set. Note that we developed another plugin that shows a detailed view of the capabilities of the container.
- PID - Container process PID.

- Effective UID of the processes - helps to detect Docker containers running as root which is a common misconfiguration.

inspect-caps

The `inspect-caps` option prints a list of running containers (taken from the `ps` option) and displays a list of containers' capabilities by their names. Each process in Linux OS has its Effective capabilities saved as a bits sequence where each bit in it is a flag for each capability. We use our improved version of linux.pslist to extract this value from sub struct of the task struct.

This plugin can help an investigator to focus on containers with dangerous capabilities, detect misconfigurations and perform damage assessment.

inspect-mounts

The `inspect-mounts` option prints a list of non-default mounts. This option is heavily based on our new version of linux.mount plugin. It walks on each mount namespace for each container process and extracts mount points from it.

This plugin doesn't show all of the container's mounts but only the non-default ones.

This plugin can help an investigator to focus on containers with dangerous mounts such as mount with write permissions to */etc* or other important os directories. It can also find the common yet dangerous mount of */var/run/docker.sock* or */run/docker.sock*.

inspect-networks

The `inspect-networks` option prints a list of Docker networks, recognized by their network segments, a 16 bit long ipv4 segment. This option is heavily based on our improved version of linux.ifconfig and its namespaces awareness. The option walks on the networking interfaces in the kernel, extract Docker-related interfaces by their MAC address, match between containers' net namespace and the interface's net ns, and group together containers by the same network segment.

Why we should win the contest?

As researchers that use the Volatility framework for memory analysis on a regular basis, we have always been bothered by the inability to extract any information about **containers** and to even distinguish between a regular application and one running inside a container.

This led us to the decision that a **dedicated plugin** (or set of plugins) needs to be developed, that can provide an investigator valuable information regarding containers running on the system, the relations between them, their usage of the host's filesystem, and their context within the **container orchestration** software.

Upon constructing the specification of the plugin, we realized that the Volatility 3 framework currently misses plugins and features, that were present in Volatility 2, that are necessary for our plugin.

This led us to **port** and **rewrite** these plugins to Volatility 3. This process took a lot of work, mainly research, because we felt the responsibility to produce **high-quality** and **well-documented** code. The result was a near-complete re-write of these plugins, adding our own **improved logic**, which required many hours of kernel research.

In addition to rewriting plugins from Volatility 2, awareness of **Linux namespaces**, which is a major feature of the Linux kernel for at least 10 years, was added to them. For example, the mount plugin now extracts mounts from the context of a specified process, which may belong to a different mount namespace than other processes and as such have different mounts. Linux namespaces are an **integral part** of container technology on Linux and are what **provide the isolation** of a container's environment from the rest of the system.

As a final note, we put a **great effort** into making sure the plugins we developed work in various scenarios, including support for many kernel versions (as old as 2.6.32), even in places where the corresponding Volatility 2 plugins did not support them.

We hope the plugins we provide will be useful for many people in the memory forensics community and will help advance the Volatility framework to the brave new world of container security.